UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ÉVALUATION DES SYSTÈMES DE GESTION DE RÈGLES ET DE FLUX DE TRAVAIL POUR LA CHAIRE DE LOGICIEL LIBRE - FINANCE SOCIALE ET SOLIDAIRE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

JIHED AMINE MAARREF

NOVEMBRE 2010

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Ce mémoire est dédié à ma mère, Zeineb Ghozlani, l'être qui m'est le plus cher au monde, ma meilleure amie et un modèle de courage, de gentillesse et de générosité.

Il est également dédié à mon père, Nejib Maaref, qui s'est toujours démené pour nous offrir les meilleures conditions de travail et qui s'est toujours soucié du bien-être de ses enfants.

Vous êtes les seules personnes qui m'aiment et que j'aime d'un amour absolu et inconditionnel. Je ne vous remercierai jamais assez.

Par ailleurs, je remercie Monsieur Louis Martin pour le respect et la confiance qu'il m'a témoignés ainsi que pour son soutien financier.

Je remercie également les membres du jury, Mr Norman Seguin et Mr Ivan Maffezini, pour avoir manifesté de l'intérêt pour mon mémoire et pour avoir consacré du temps à le lire.

Enfin, je remercie le personnel et les enseignants de l'école primaire de Tazarka, du lycée Pilote de Tunis, de l'Institut Supérieur d'Informatique de l'Ariana, de l'Université Libre de Tunis et de l'Université du Québec à Montréal pour avoir assuré mon éducation scolaire.

TABLE DES MATIÈRES

LISTE DES TABLEAUX xiii
LISTE DES FIGURES xv
RÉSUMÉ xxv
INTRODUCTION 1
CHAPITRE I
LA FINANCE SOCIALE ET SOLIDAIRE ET LE LOGICIEL LIBRE 5
1.1 Économie et finance sociale et solidaire 5
1.1.1 Principes fondateurs 5
1.1.2 Illustrations concrètes 6
1.2 Le logiciel libre 7
1.2.1 Principes fondateurs 7
1.2.2 Dimension éthique 8
1.2.3 Adoption du logiciel libre 9
1.3 La Chaire de logiciel libre - Finance sociale et solidaire 10
CHAPITRE II
LES SYSTÈMES DE GESTION DE RÈGLES ET DE FLUX DE TRAVAIL . . 11
2.1 Les moteurs de règles 11
2.1.1 Les règles d'affaires 11
2.1.2 Les moteurs de règles 11
2.1.3 Appariement de modèles (Pattern Matching) 15
2.1.4 Standards de représentation des règles d'affaires 18
2.1.5 Différence entre moteur de règles et système de gestion de règles . . 21
2.1.6 Apports des moteurs de règles 22
2.1.7 Quand opter pour un moteur de règles ? 23
2.2 Les moteurs de flux de travail 25
2.2.1 Processus d'affaires (Business Process) 25

2.2.2 Flux de travail 27
2.2.3 Définition des moteurs de flux de travail 27
2.2.4 Langages de définition des processus d'affaires 29
2.2.5 Différence entre moteur de flux de travail et système de gestion de flux de travail 37
2.2.6 Apports des moteurs de flux de travail 38
2.3 Interactions entre moteurs de règles et moteurs de flux de travail 40
2.3.1 Nécessité de séparer les règles des processus d'affaires 40
2.3.2 Scénarios d'interaction entre moteurs de règles et moteurs de flux de travail 41
CHAPITRE III
CRITÈRES D'ÉVALUATION DES MOTEURS DE RÈGLES 43
3.1 Critères fonctionnels 44
3.1.1 Langage de définition des règles 44
3.1.2 Fonctions facilitant l'expression des règles 47
3.1.3 Accessibilité aux experts métier 49
3.1.4 Le moteur de règles 50
3.1.5 Outils de vérification 53
3.1.6 Le stockage des règles 55
3.2 Critères non fonctionnels 57
3.2.1 Performance et coûts en ressources 58
3.2.2 Prise en main 60
3.2.3 Évaluation sur l'aspect logiciel libre 61
3.3 Pondération des critères d'évaluation selon l'environnement 63
3.3.1 Critères invariablement importants 64
3.3.2 Scénario 1 : Règles volumineuses et sophistiquées 64
3.3.3 Scénario 2 : Variété des utilisateurs 65
3.3.4 Scénario 3 : Existence de compétences informatiques internes importantes 66
3.3.5 Scénario 4 : Implication des utilisateurs métier 67
3.4 Grilles d'évaluation des moteurs de règles 67

CHAPITRE IV
CRITÈRES D'ÉVALUATION POUR LES MOTEURS DE FLUX DE TRAVAIL 71

4.1 Les patrons de flux de travail 71

4.2 Critères fonctionnels 74

4.2.1 Langage de définition des processus 74

4.2.2 Contrôle du flux de travail 75

4.2.3 Distribution des items de travail 79

4.2.4 Représentation et utilisation des données dans le flux 83

4.2.5 Accessibilité aux experts métier 86

4.2.6 Le moteur de flux de travail 88

4.2.7 Outils de vérification 89

4.2.8 Stockage des processus 91

4.3 Critères non fonctionnels 92

4.3.1 Conformité aux standards 92

4.3.2 Performance et coûts en ressources 93

4.3.3 Prise en main 94

4.3.4 Évaluation sur l'aspect logiciel libre 94

4.4 Pondération des critères d'évaluation selon l'environnement 94

4.4.1 Critères invariablement importants 95

4.4.2 Scénario 1 : Collaboration conséquente entre le moteur de flux et l'environnement externe 95

4.4.3 Scénario 2 : Agilité, besoin d'adaptation au changement et d'autonomie des composants 96

4.4.4 Scénario 3 : Organisme de moyenne ou grande taille avec des flux complexes et un département informatique interne 97

4.4.5 Scénario 4 : Organisme limité en ressources logistiques et en personnel informatique 98

4.5 Grille d'évaluation des moteurs de flux de travail 98

CHAPITRE V
ÉVALUATION DES MOTEURS DE RÈGLES 103

5.1 Présentation des produits à évaluer 103

5.1.1 Drools 105
5.1.2 CLIPS 105
5.1.3 Open Rules 106
5.2 Langage de définition des règles 107
5.2.1 Prise en charge de plusieurs moyens d'expression des règles 107
5.2.2 Expressivité du langage de définition des règles 111
5.2.3 Prise en charge de l'internationalisation 115
5.2.4 Compréhension du modèle de données 117
5.3 Fonctions facilitant l'expression des règles 122
5.3.1 Modèles de règles 122
5.3.2 Procédures et algorithmes 125
5.3.3 Flux de règles 129
5.3.4 Tables et arbres de décision 135
5.4 Accessibilité aux experts métier 138
5.4.1 Accessibilité du langage aux non informaticiens 138
5.4.2 Abstraction des détails d'implémentation 140
5.4.3 Interfaces pour non informaticiens 142
5.5 Le moteur de règles 151
5.5.1 Stratégies du moteur d'inférence 151
5.5.2 Explication du raisonnement et gestion de l'incertitude 154
5.5.3 Intégration à l'environnement 156
5.5.4 Changements dynamiques 160
5.6 Outils de vérification des règles 162
5.6.1 Vérification de la syntaxe des règles 162
5.6.2 Stratégies de résolution de conflit 165
5.6.3 Débogage 167
5.6.4 Test des règles 170
5.6.5 Journalisation et audit 176
5.7 Le stockage des règles 181

5.7.1 Versionnage des règles 181
5.7.2 Hiérarchisation des règles 184
5.7.3 Définition de méta-données 187
5.7.4 Recherche par critères 189
5.7.5 Administration des dépôts 192
5.8 Performance et coût 195
5.8.1 Performance des moteurs de règles 195
5.8.2 Coûts en ressources 199
5.9 Prise en main 202
5.9.1 Documentation 202
5.9.2 Convivialité 204
5.9.3 Courbe d'apprentissage 206
5.10 Évaluation sur l'aspect logiciel libre 208
5.10.1 Communauté 208
5.10.2 Publications 210
5.10.3 Longévité 211
5.10.4 Licence 213
5.10.5 Assistance communautaire 214
5.10.6 Assistance professionnelle 216
5.11 Résultats des évaluations 217
5.11.1 Scénario 1 217
5.11.2 Scénario 2 219
5.11.3 Scénario 3 221
5.11.4 Scénario 4 223
5.11.5 Bilan 225
CHAPITRE VI
ÉVALUATION DES MOTEURS DE FLUX DE TRAVAIL 227
6.1 Présentation des produits à évaluer 227
6.1.1 jBPM 229

6.1.2 Bonita Open Solution 230
6.1.3 Enhydra Shark 230
6.2 Langage de définition des processus 231
6.2.1 Prise en charge de plusieurs moyens d'expression 231
6.2.2 Prise en charge de l'internationalisation 240
6.2.3 Compréhension du modèle de données 242
6.3 Contrôle du flux de travail 245
6.3.1 Contrôle élémentaire du flux 245
6.3.2 Contrôle avancé du flux 250
6.3.3 Contrôle selon l'état du processus 256
6.3.4 Itération dans le flux 259
6.3.5 Annulation d'éléments dans le flux 262
6.3.6 Gestion d'instances multiples 265
6.3.7 Démarrage déclenché par des signaux externes 268
6.4 Distribution des items de travail 270
6.4.1 Distribution à la conception 270
6.4.2 Allocation des ressources par le système 277
6.4.3 Engagement des ressources à exécuter des tâches 280
6.4.4 Modification des attributions de tâches aux ressources 283
6.5 Représentation et utilisation des données dans le flux 290
6.5.1 Définition des données pour un flux selon plusieurs contextes 290
6.5.2 Transfert de données entre composants d'un processus 294
6.5.3 Transfert de données entre processus et environnement 297
6.5.4 Prise en charge de divers mécanismes de transfert 300
6.5.5 Utilisation des données comme conditions 303
6.6 Accessibilité aux experts métier 305
6.6.1 Définition graphique des processus 305
6.6.2 Définition assistée des activités 310
6.6.3 Interfaces accessibles web et optimisées pour les non informaticiens . 315

6.6.4 Abstraction des détails d'implémentation 321
6.6.5 Modèles de processus 323
6.7 Le moteur de flux de travail 324
6.7.1 Intégration à l'environnement 324
6.7.2 Changements dynamiques 328
6.8 Outils de vérification 332
6.8.1 Vérification de la syntaxe 332
6.8.2 Test 336
6.8.3 Débogage 340
6.8.4 Journalisation et audit 343
6.9 Stockage des processus 348
6.9.1 Versionnage 348
6.9.2 Importation/Exportation de processus 351
6.10 Conformité aux standards 353
6.11 Performance et coût 355
6.11.1 Performance des moteurs de flux 355
6.11.2 Coûts en ressources matérielles 360
6.12 Prise en main 362
6.12.1 Documentation 362
6.12.2 Convivialité 366
6.12.3 Courbe d'apprentissage 368
6.13 Évaluation sur l'aspect logiciel libre 370
6.13.1 Communauté 370
6.13.2 Publications 373
6.13.3 Longévité 376
6.13.4 Licence 377
6.13.5 Aide communautaire 380
6.13.6 Assistance professionnelle 381
6.14 Résultats des évaluations 384

6.14.1 Scénario 1 384
6.14.2 Scénario 2 386
6.14.3 Scénario 3 388
6.14.4 Scénario 4 391
6.14.5 Bilan 393
CHAPITRE VII
RECOMMANDATIONS 395
7.1 Points forts et points faibles des produits évalués 395
7.1.1 Moteur de règles 395
7.1.2 Moteur de flux 397
7.2 Solution recommandée pour la gestion de règles et de processus d'affaires . 400
7.2.1 Apports de JBoss Drools 400
7.2.2 Apports de Bonita Open Solution 402
7.2.3 Collaboration entre les deux systèmes 403
CONCLUSION 405
BIBLIOGRAPHIE 407

LISTE DES TABLEAUX

2.1 Exemple de règle d'affaire 12

3.1 Grille d'évaluation des moteurs de règles 70

4.1 Extrait des évaluations (Wohed et al., 2009b) 73

4.2 Grille d'évaluation des moteurs de flux 101

5.1 Évaluation des moteurs de règles selon S1 219

5.2 Évaluation des moteurs de règles selon S2 221

5.3 Évaluation des moteurs de règles selon S3 223

5.4 Évaluation des moteurs de règles selon S4 225

6.2 Évaluation des moteurs de flux selon S1 386

6.3 Évaluation des moteurs de flux selon S2 388

6.4 Évaluation des moteurs de flux selon S3 391

6.5 Évaluation des moteurs de flux selon S4 393

LISTE DES FIGURES

2.1 Composants d'un moteur de règles 12

2.2 Flux de travail d'un moteur d'inférence [d'après (Cirstea et Kirchner, 2004)] 13

2.3 Illustration de l'appariement de modèles de Rete 17

2.4 Table décisionnelle 20

2.5 Arbre de décision 20

2.6 Terminologie des flux de travail (Coalition, 1999) 26

2.7 Listes et unités de travail (Coalition, 1999) 28

2.8 Modèle de référence du flux de travail (Coalition, 1999) 29

2.9 Exemples d'éléments graphiques BPMN 31

2.10 Notation BPMN d'une demande de prêt simple 32

2.11 Structure d'un fichier XPDL 33

2.12 Implémentation d'un diagramme BPMN en XPDL (White, 2003) 34

5.1 Table décisionnelle de règles de salutation 110

5.2 Table d'une BD relationnelle des règles de salutation 111

5.3 Règle exprimée avec OpenRules 114

5.4 Données définies dans un tableur 120

5.5 Import de données Java, XML et XLS avec OpenRules 121

5.6 Modèle de règles OpenRules 124

5.7 Déclaration d'une méthode dans OpenRules 128

5.8 Flux Drools Flow de traitement d'une demande d'emprunt (Bali, 2009) 130

5.9 Définition d'une fenêtre OpenRules 134

5.10 Définition de l'étape précédente et suivante de chaque fenêtre 135

5.11 Règles et fenêtre à déclencher à chaque étape 135

5.12 Table décisionnelle de règles de taux d'intérêt (Bali, 2009) 136

5.13 Les détails techniques de la table décisionnelle sont cachés 140

5.14 Détails techniques exposés 141

5.15 Détails techniques cachés 142

5.16 Auto-complétion des règles Drools 143

5.17 Assistant de définition de règles Drools 143

5.18 Contrôle des permissions des utilisateurs de Guvnor 144

5.19 Présentation des bases de connaissances par Guvnor 145

5.20 Assistant de définition de faits Guvnor 146

5.21 Tables décisionnelles web Guvnor 146

5.22 Assistant d'affectation d'une priorité à une règle 147

5.23 Gestion de la liste de classes 148

5.24 Création d'un projet OpenRules avec le plugiciel *eclipse* 149

5.25 Choix d'une valeur dans une liste prédéfinie 149

5.26 Interface web d'exécution d'une règle 150

5.27 Validation des faits saisis 151

5.28 Vérificateur syntaxique Drools 163

5.29 Détection des erreurs dans les tableurs 163

5.30 Détection d'erreurs dans les tables décisionnelles 164

5.31 Ajout d'un point d'arrêt dans une règle 167

5.32 Débogage Drools d'une application 167

5.33 Représentation de l'activité en cours dans un flux Drools 168

5.34 Vue *eclipse* des instances de processus Drools 168

5.35 Scénario de test dans Guvnor 172

5.36 Suite de tests dans Guvnor 173

5.37 Analyse d'un paquetage de règles par Drools-Verifier 177

5.38 Vue *eclipse* du journal d'audit Drools 178

5.39 Entrées dans un journal OpenRules 180

5.40 Visualisation d'une version précédente d'une règle 182

5.41 Changement de l'état d'une règle 182

5.42 Accès à un dépôt avec versionnage à partir de tableurs 184

5.43 Catégories Guvnor 185

5.44 Hiérarchisation en paquetages Drools 185

5.45 Hiérarchisation en dossiers OpenRules 186

5.46 Méta-données d'une règle 188

5.47 Recherche dans un dépôt 190

5.48 Journal des actions des utilisateurs du dépôt Guvnor 193

5.49 Accès WebDAV à un dépôt Guvnor à partir d'*eclipse* 194

6.1 Éditeur d'expressions Bonita 234

6.2 Extrait de la liste des connecteurs prédéfinis par Bonita 235

6.3 Connecteur Java exécutant du code Python 236

6.4 Choix de l'étape de déclenchement du connecteur 237

6.5 Ajout de la prise en charge d'une langue par un connecteur Bonita . . . 241

6.6 Contrôles élémentaires de flux avec Bonita 247

6.7 Séquence de tâches avec ES 248

6.8 Divergence parallèle et synchronisation avec ES 248

6.9 Choix exclusif et convergence simple avec ES 249

6.10 Choix multiples avec Bonita 252

6.11 Choix et convergence multiples avec ES 253

6.12 Choix multiples et convergence synchronisante avec ES 254

6.13 Discriminateur avec ES 256

6.14 Boucles avec Bonita 260

6.15 Récursivité avec ES 261

6.16 Annulation d'un ensemble d'instances de processus actives 263

6.17 Annulation d'une tâche à partir de l'interface graphique ES 265

6.18 Configuration de l'instanciateur pour une étape à instances multiples . . 266

6.19 Instances multiples connues à la conception avec ES 267

6.20 Démarrage déclenché par un message de type throw 269

6.21 Gestion des utilisateurs dans User XP 274

6.22 Correspondance entre participants et utilisateurs 276

6.23 Gestion des groupes avec ES 276

6.24 L'utilisateur authentifié est capable de réclamer une tâche 281

6.25 Gestion des tâches par la ressource avec Bonita 281

6.26 Gestion des tâches par la ressource avec ES 282

6.27 Réassigner une tâche à une autre ressource 286

6.28 Se désengager d'une tâche 286

6.29 Délégation d'un item de travail avec ES 288

6.30 Définition d'une variable de processus ES 293

6.31 Envoi de données dans un message 295

6.32 Correspondance entre résultats du connecteur et variables du processus 299

6.33 Précondition et postcondition pour l'étape Vérifier le compte 304

6.34 Définition graphique de flux avec le plugiciel *eclipse* 306

6.35 Application web Signavio de modélisation de flux 306

6.36 L'éditeur graphique liste les types d'éléments possibles 307

6.37 Palette de composants de l'éditeur TWE 308

6.38 Fenêtre popup des propriétés d'un élément 309

6.39 Vue *Navigator* de l'éditeur TWE 310

6.40 Formulaire généré pour la console jBPM 311

6.41 Assistant de configuration d'un dépôt LDAP 312

6.42 Éditeur de formulaires Bonita 313

6.43 Validateurs de formulaires Bonita 313

6.44 Assistant de définition d'un agent 314

6.45 Assistant de définition des participants 314

6.46 Assistant de définition d'un paramètre formel 315

6.47 Gestion des processus dans la console jBPM 316

6.48 Boîte de réception pour une ressource 317

6.49 Actions possibles pour une tâche 318

6.50 Fil des tâches d'une instance de processus 318

6.51 Version Swing de l'application d'administration 319

6.52 Déploiement à chaud d'un processus 330

6.53 Mise à jour de la définition d'un processus 331

6.54 Vérification des diagrammes Bonita 333

6.55 Coloration syntaxique du code XML généré à partir d'un diagramme Bonita334

6.56 Liste des problèmes de validation 334

6.57 Coloration syntaxique avec la vue XPDL de TWE 335

6.58 Test du connecteur Drools 338

6.59 Test des expressions Groovy 338

6.60 Suivi de l'exécution du processus 341

6.61 Affichage des valeurs des variables pendant l'exécution 341

6.62 Requêtes de sélection pendant l'exécution de processus 343

6.63 Génération de rapports dans la console jBPM 345

6.64 Rapport d'audit User XP 347

6.65 Versionnage Bonita 350

6.66 Versionnage des paquetages et des processus ES 350

6.67 Importation de définitions de processus dans Bonita 352

LISTE DES ACRONYMES

AI2L	Association Internationale du Logiciel Libre
ANSI	American National Standards Institute
API	Application Programming Interface
BPD	Business Process Diagram
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
BSD	Berkeley Software Distribution
CLIPS	C Language Integrated Production System
FLOSS	Free / Libre Open Source Software
FOSS	Free Open Source Software
FSF	Free Software Foundation
GNU	GNU's Not Unix
GPL	GNU General Public License
GWT	Google Web Toolkit
IDE	Integrated Development Environment
Java EE	Java Enterprise Edition
JCP	Java Community Process
jPDL	Java Process Definition Language
JSR	Java Specification Request
JVM	Java Virtual Machine
KB	Knowledge Base
LGPL	GNU Lesser General Public License
LHS	Left Hand Side
MIT	Massachussets Institute of Technology
MOF	Meta Object Facility

OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
POJO	Plain Old Java Object
PRR	Production Rule Representation
PVM	Process Virtual Machine
REST	Representational State Transfer
RHS	Right Hand Side
RDF	Resource Description Framework
RIF	Rules Interchange Format
SGBD	Système de Gestion de Base de Données
SOAP	Simple Object Access Protocol
SVBR	Semantics and Vocabulary of Business Rules
SVN	SubVersioN
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WARIA	Workflow And Reengineering International Association
WfMC	Workflow Management Coalition
WME	Working Memory Element
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Definition Language
XMI	XML Metadata Interchange
XML	eXtended Markup Language
XPDL	XML Process Definition Language

RÉSUMÉ

Ce mémoire introduit les systèmes de gestion de règles et de flux de travail. En faisant la synthèse de critères d'évaluation établis dans la littérature pour ces deux types de produits et en dégageant des critères spécifiques aux besoins des organismes de finance sociale et solidaire, le mémoire établit une grille d'évaluation pour les systèmes de gestion de règles et une seconde pour les systèmes de gestion de flux de travail. Les logiciels JBoss Drools, CLIPS et OpenRules sont evalués selon la première grille. Les logiciels JBoss jBPM, Bonita Open Solution et Enhydra Shark sont évalués selon la seconde. À la lumière des résultats obtenus, JBoss Drools et Bonita Open Solution sont recommandés comme étant les solutions libres les plus adaptées pour gérer respectivement les règles et les processus d'affaires des organismes de finance sociale et solidaire.

ABSTRACT

This Master Thesis introduces rule and workflow management systems. Evaluation criteria established in litterature for both types of products are synthesized and combined with specific criteria for social and solidary finance organizations, resulting in two evaluation grids, one for business rules management systems and a second one for worflow management systems. JBoss Drools, CLIPS and OpenRules are evaluated with the first grid. JBoss jBPM, Bonita Open Solution and Enhydra Shark are evaluated with the second one. In the light of the obtained results, JBoss Drools and Bonita Open Solution are recommended as the most adequate free solutions for social and solidary financial organizations to respectively manage their business rules and their workflows.

INTRODUCTION

Problématique

Les organismes de finance solidaire et sociale, notamment dans les pays en voie de développement, doivent se munir de systèmes informatiques assurant leur survie dans un milieu compétitif et très dynamique. Cependant, la majorité des produits existants pour le secteur financier sont propriétaires, développés en interne dans des sociétés à gros budget ou commercialisés à des prix très élevés. De plus, les logiques métier qu'ils implémentent sont orientées vers une maximisation des profits et ne sont donc pas en accord avec la philosophie des entreprises de finance solidaire qui portent une responsabilité sociale. La Chaire de logiciel libre - Finance sociale et solidaire a pour objectif, entre autres, de concevoir et développer un système informatique libre, en accord avec les moyens et les besoins des organismes de finance solidaire et sociale.

Ce mémoire vise des problématiques spécifiques à la gestion des règles d'affaires et des flux de travail dans le cadre de la chaire. D'abord, les règles d'affaires évoluent rapidement dans le domaine financier. Elles devraient donc être définies en dehors du code applicatif du système informatique utilisé. Leur évaluation et l'exécution d'actions conséquentes à leur satisfaction devraient être performantes. Leur représentation devrait aussi être intuitive afin de faciliter leur définition et leur modification. Les gens du métier auront ainsi un meilleur contrôle sur leur système informatique et pourront l'administrer en limitant l'intervention d'experts informatiques. Ceci est d'autant plus souhaitable que les ressources financières et humaines sont limitées dans l'économie solidaire. D'autre part, les entreprises concernées ont souvent des effectifs de petite ou moyenne taille et ne disposent généralement pas de systèmes automatisés de gestion des processus de travail qu'utilisent des entreprises de finance à but capitaliste. L'application à développer par la chaire aura à rationaliser le flux de travail en définissant les processus et les acteurs

au sein de l'organisme de finance solidaire et en automatisant sa gestion. La formalisation des opérations permettra une meilleure interaction avec les clients et améliorera l'efficacité et le rendement de ces structures qui seront ainsi plus compétitives et plus rentables. De plus, les organismes de finance solidaire se trouvent dans des environnements différents avec des politiques internes et des contraintes différentes. Il est donc souhaitable de réduire autant que possible les coûts de personnalisation en offrant une plateforme flexible qui puisse être aisément adaptée selon les besoins spécifiques d'un organisme donné. Enfin, les solutions logicielles à adopter doivent respecter les budgets généralement restreints de ces organismes, leur permettre une indépendance totale de quelque fournisseur privé de produits logiciels que ce soit et leur offrir le moyen de modifier et de partager le code applicatif de la solution utilisée. Les systèmes de gestion de règles et de flux de travail libres et à code ouvert (Open Source) constituent d'éventuelles réponses à ces besoins. Jusqu'à présent, ce type de solutions a été principalement utilisé dans des entreprises capitalistes disposant de leurs propres moyens de personnalisation et de maintenance. Une solution globale, libre et axée sur la facilité de réutilisation et l'évolution ne s'est pas encore imposée dans les organismes de finance solidaire et sociale, qui relèvent les défis qui leurs sont posés sans bénéficier de ces nouvelles technologies.

Objectifs

Dans ce mémoire, on tentera de rassembler les critères d'évaluation des systèmes de gestion de règles et de flux de travail présents dans la littérature. D'autre part, on dégagera les exigences propres aux organismes de finance sociale et solidaire en matière d'administration des règles d'affaires et des flux de travail. De la compilation de l'ensemble de ces critères résulteront deux grilles d'évaluation personnalisées qu'on appliquera à une sélection de produits libres disponibles sur le marché. Au bout de cette évaluation, on espère déterminer une solution optimale de systèmes de gestion de règles et de flux de travail qu'on recommandera à la chaire pour le développement de son système informatique.

i. Définition des exigences de gestion de règles et de formalisation de processus

Les exigences sur lesquelles se basera l'évaluation des produits existants sont à la fois des exigences techniques recensées dans la littérature mais également des exigences spécifiques à la finance sociale et solidaire.

En ce qui concerne les critères établis, plusieurs études, articles et livres recensent des patrons d'évaluation des systèmes de gestion de flux [1]. Des bancs d'essais établis dans l'industrie tel que *WaltzDB* et *MissManers* permettent quant à eux d'évaluer les performances des moteurs de règles d'affaires. L'objectif de cette étape est d'assimiler les résultats des différentes études disponibles sur le sujet.

Concernant les besoins spécifiques à l'environnement de la finance solidaire et sociale, on déterminera les propriétés et les contraintes liées au contexte. On dégagera ainsi des qualités communes que devront posséder les systèmes de gestion de règles et de flux de travail à sélectionner pour la chaire.

Au delà de partager globalement le même statut et les mêmes objectifs, les organismes peuvent avoir des volumes de règles et de processus d'affaires différents, des compositions de personnel différentes ou encore des services de natures différentes. C'est pour cela qu'on définira plusieurs environnements types selon lesquels on pondérera l'importance des critères établis précédemment. Il se pourra, par exemple, qu'une certaine solution soit recommandée pour un profil donné d'organismes, mais qu'une autre solution réponde mieux aux besoins d'un autre type d'organismes.

ii. Analyse des systèmes de gestion de règles et de flux de travail existants

Parmi le large éventail de produits à code ouvert, on sélectionnera les trois systèmes de gestion de règles et les trois systèmes de gestion de flux les plus activement développés, les plus aboutis et les plus susceptibles d'être utilisés dans un environnement de production. Chaque produit sera analysé selon tous les critères établis pour sa catégorie (règles ou flux de travail). Une note de 0, 1 ou 2 sera attribuée à un produit s'il ne satisfait

---

1. (Wohed et al., 2009a), (van der Aalst et van Hee, 2002)

pas un critère, s'il y répond partiellement ou s'il le comble totalement. Les notes seront ensuite pondérées selon les types d'environnements définis.

iii. Recommandation d'une solution finale pour l'application financière de la chaire

À la lumière des résultats obtenus, cette dernière partie du mémoire récapitulera les points forts et les points faibles des produits analysés. Pour chacun d'entre eux, on dégagera les cas de figures où l'utilisation du logiciel est pertinente. Enfin, une recommandation finale d'un système de gestion de règle et d'un système de gestion de flux de travail sera émise. Les bénéfices qu'en tireront les organismes de finance sociale et solidaire seront également énumérés.

Pertinence

Les organismes de finance solidaire et sociale n'ont pas adopté dans une large proportion des systèmes de gestion de règles et de flux de travail, du fait qu'ils ne peuvent se permettre de payer des équipes onéreuses pour développer ou personnaliser selon leurs besoins une solution utilisant ces technologies. La disponibilité de systèmes de gestion de règles et de flux de travail libres, à code ouvert, sélectionnés selon les besoins des entreprises de finance sociale et solidaire, faciliterait l'adoption de ces technologies dans de tels organismes. Ces derniers pourront ainsi être plus flexibles face aux changements de leurs règles d'affaires et les maintenir plus facilement. Ils rationaliseront leurs processus et en assureront un meilleur suivi, améliorant ainsi leur efficacité et leur compétitivité. L'étude sort donc les moteurs de règles et de flux de travail de la sphère des corporations et offre une solution libre et globale pour des entreprises responsables socialement. L'adoption de telles technologies constitue une étape dans la démocratisation des nouvelles technologies.

CHAPITRE I

LA FINANCE SOCIALE ET SOLIDAIRE ET LE LOGICIEL LIBRE

Ce premier chapitre introduit la notion de finance sociale et solidaire et celle de logiciel libre. Il s'arrête sur les principes fondateurs des deux mouvements et en cite quelques implémentations concrètes.

1.1 Économie et finance sociale et solidaire

1.1.1 Principes fondateurs

La philosophe et novéliste Ayn Rand disait : « L'argent n'est qu'un outil, il vous conduira où vous voudrez, mais il ne vous remplacera pas en tant que conducteur[1] » [Notre traduction]. En ces temps de crise financière, il semble judicieux de promouvoir des systèmes financiers alternatifs, guidés par les intérêts collectifs et non par les intérêts d'un ensemble restreint de spéculateurs et détenteurs de capitaux. La finance, et plus généralement l'économie sociale et solidaire sont nées de cette aspiration.
L'économie sociale et solidaire a été définie comme : « un ensemble d'initiatives économiques à finalité sociale qui participent à la construction d'une nouvelle façon de vivre et de penser l'économie à travers des dizaines de milliers de projets dans les pays du Nord comme du Sud. Elle place la personne humaine au centre du développement économique et social. La solidarité en économie repose sur un projet tout à la fois économique, politique et social, qui entraîne une nouvelle manière de faire de la politique et d'établir

1. Ayn Rand, *Atlas Shrugged*, Random House Publishing, 1957

les relations humaines sur la base du consensus et de l'agir citoyen[2] ».

La finance sociale et solidaire applique les principes de responsabilité citoyenne dans le secteur de la finance. Une publication du Conseil de l'Europe en recense trois formes principales : la finance éthique, la finance solidaire par redistribution des gains et la finance d'investissements solidaires (d'Orfeuil, 2005). La finance éthique consiste à désigner, à travers des agences spécialisées, les entreprises ayant une politique socialement ou environnementalement responsable. Les épargnants et les investisseurs les récompenseront en y faisant leurs investissements. Deuxièmement, la redistribution des gains consiste à faire don d'une partie des revenus d'une institution à des associations bénévoles. Enfin, la finance d'investissements solidaires crée des produits qui allient les objectifs financiers traditionnels avec des objectifs sociaux, locaux et environnementaux. Les types de produits sont divers et les autorités publiques s'engagent de plus en plus à en faire la promotion. Plus généralement, la finance solidaire et l'économie sociale pensent l'économie comme un *moyen* configurable « en fonction des choix politiques et sociaux faits en amont »(Gendron et Bourque, 2003).

1.1.2 Illustrations concrètes

Parmi les entreprises francophones mettant en pratique les principes de l'économie et de la finance sociale et solidaire, nous pouvons citer :

- Fondaction[3] : Fonds des travailleurs de la confédération des Syndicats Nationaux du Québec, Fondaction offre une épargne retraite avantageuse pour les travailleurs syndiqués. Ses investissements se portent majoritairement sur des entreprises québécoises afin de favoriser le maintient et la création d'emplois dans la province.
- Caisse d'économie solidaire Desjardins[4] : Offre des placements à rendement social dans des projets de logements sociaux, de commerce équitable ou d'autres projets

2. 2e rencontre internationale sur la globalisation de la solidarité (Québec, octobre 2001)

3. http ://www.fondaction.com (dernière consultation le 09-11-10)

4. http ://www.cecosol.coop/ (dernière consultation le 09-11-10)

communautaires. L'institution propose également des taux d'intérêt avantageux à l'achat de véhicules éco-énergétiques ainsi que des prêts hypothécaires pour les travaux immobiliers respectueux de l'environnement. La caisse s'implique également dans le développement solidaire international.

- Macif[5] : Premier assureur automobile et premier assureur d'habitation en France avec 3 millions de maisons assurées, 528 bureaux locaux et 33 centres d'appel ; la Macif n'en est pas moins socialement responsable. En effet, elle n'appartient pas à des actionnaires et le pouvoir y revient aux sociétaires, représentés par des délégués. En plus des rapports financiers traditionnels, elle publie chaque année un rapport mutualiste où elle rend compte de son implication sociale et de son engagement pour un mode de gouvernance participatif[6].

Plusieurs autres entreprises prennent part, d'une manière ou d'une autre, à l'économie sociale et solidaire.

Tout comme l'économie sociale et solidaire s'est posée comme une alternative au capitalisme pur ; les logiciels libres ont contesté la philosophie des logiciels propriétaires.

1.2 Le logiciel libre

1.2.1 Principes fondateurs

Dans les années 80, la norme était de cacher le code source des logiciels et de restreindre leur utilisation avec des licences. En 1983, Richard Stallman crée le projet *GNU* et la *Free Software Foundation* qui ont pour buts respectifs de développer un système d'exploitation Unix composé de logiciels libres et de formaliser la notion de logiciel libre.

La définition formelle du logiciel libre par la FSF[7] établit quatre niveaux de liberté :

---

5. http ://www.macif.fr/ (dernière consultation le 09-11-10)

6. Projet mutualiste du groupe Macif, Congrès de Chantilly, 18 Juin 2005

7. (Stallman, 2002), Pages 43-45

- Liberté d'exécuter un programme, indépendamment du but.
- Liberté d'étudier le fonctionnement du programme et de le modifier selon ses désirs.
- Liberté de redistribuer des copies du programme.
- Liberté d'améliorer le programme et de publier les améliorations au public, pour le bien collectif.

1.2.2 Dimension éthique

Le logiciel libre porte un message moral. Dans un essai intitulé *Why "Free Software" is better than "Open Source"*[8], Richard Stallman explique que le logiciel libre ne se limite pas à une obligation technique d'ouverture du code, mais porte des principes d'entraide communautaire et de partage de connaissances. Il cite l'exemple d'entreprises qui utilisent le label "Open Source" pour gagner la sympathie de la communauté, mais qui offrent une partie de leurs produits en code ouvert pour mieux amener les clients potentiels à consommer d'autres produits propriétaires à code fermé. Stallman pense que le mouvement du logiciel libre doit cultiver chez les utilisateurs la conscience de leur droit à la liberté afin qu'ils refusent de telles pratiques. Ainsi, le logiciel libre est plus qu'un modèle technique de publication de logiciels ; c'est surtout une philosophie et des principes. Les libertés qu'il promeut visent à éliminer la logique de parts de marché et les situations de monopole, à créer des standards ouverts et à partager librement du code de qualité.

La philosophie du logiciel libre est parfois considérée comme techniquement contraignante voir impossible à implémenter de manière absolue. C'est pourquoi il existe plusieurs licences libres qui diffèrent dans certains détails afin d'accommoder des besoins divers. Parmi les licences les plus utilisées, on peut citer la licence GPL (GNU General Public License), la licence Apache, la licence MIT (Massachussets Institute of Technology) et la licence BSD (Berkeley Software Distribution). Une même licence peut posséder plusieurs versions. Les différences entre licences résident principalement dans

8. (Stallman, 2002), Pages 57-62

la présence ou non de certaines clauses jugées nécessaires par certains et restrictives par d'autres. À titre d'exemple, les licences permissives, contrairement aux licences de type Copyleft, autorisent la réutilisation du code libre dans des projets propriétaires.

1.2.3 Adoption du logiciel libre

Les notions de logiciels libres et de logiciels à code ouvert sont unies dans le terme *FOSS* ou *FLOSS* (Free / Libre / Open Source Software). Les produits qui tombent sous cette appellation ont une place importante dans le monde informatique. À titre illustratif, le serveur web Apache était à 58.86% de parts de marché en 2007. En 2004, 67% des entreprises utilisaient des produits FLOSS (Wheeler, 2007).
(Fitzgerald, 2006) note qu'« il aurait semblé absurde, seulement quelques années en arrière, de suggérer que le pouvoir de l'industrie du logiciel propriétaire, tel que l'illustre Microsoft, pouvait être menacé par un mouvement *open source* principalement volontaire. Ce mouvement, cependant, a altéré la nature même de l'industrie logicielle. » [Notre traduction].
Au niveau du secteur public, l'indépendance vis à vis d'un fournisseur contrôlant le code source et son édition, les économies budgétaires et la qualité équivalente sinon supérieure des logiciels FLOSS par rapport aux logiciels propriétaires font que de plus en plus de gouvernements et d'institutions publiques à travers le monde choisissent de les adopter.

Le logiciel libre et l'économie sociale convergent dans plusieurs principes fondamentaux tel que l'esprit de partage et le service du bien collectif. L'idée d'offrir des solutions logicielles pour la finance sociale et solidaire qui soient en accord avec ses valeurs fondamentales s'est concrétisée dans la Chaire de logiciel libre - Finance sociale et solidaire (dorénavant appelée 'Chaire').

1.3 La Chaire de logiciel libre - Finance sociale et solidaire

La mission de la Chaire consiste d'une part à lancer une famille de logiciels libres dédiés aux services financiers solidaires et d'autre part à faire des recherches sur le développement, l'adoption et l'évolution du logiciel libre en général (Martin, 2008). La chaire est financée par l'Association Internationale du Logiciel Libre (AI2L). L'AI2L a été fondée par des acteurs de l'économie sociale francophone de premier plan, tel que Fondaction et la Macif. Dans sa charte, l'association s'engage « à soutenir la création [de logiciels libres] dans le respect des valeurs de l'économie sociale et solidaire[9]. »

Dans ce chapitre, nous avons introduit les valeurs fondamentales de la Chaire. Ce mémoire s'inscrit dans son volet développement. Plus spécifiquement, il s'agit de considérer l'utilisation de systèmes libres de gestion de règles et de flux de travail dans les entreprises de finance sociale et solidaire. La définition de ces deux types de systèmes est donnée au chapitre suivant.

9. http ://www.ai2l.org/spip/charte.php (dernière consultation le 09-11-10)

CHAPITRE II

LES SYSTÈMES DE GESTION DE RÈGLES ET DE FLUX DE TRAVAIL

Ayant introduit le logiciel libre et la finance sociale et solidaire, nous définissons dans ce chapitre les systèmes de gestion de règles et de flux de travail. Les notions de bases des deux technologies sont explicitées ainsi que leurs apports éventuels à la finance sociale et solidaire.

2.1 Les moteurs de règles

2.1.1 Les règles d'affaires

Les règles d'affaires sont des « phrases définissant ou contraignant certains aspects du domaine d'affaires » (Group, 2000). Elles sont plus généralement appelées *règles de production.*

2.1.2 Les moteurs de règles

Les moteurs de règles sont des composants logiciels enfichables qui exécutent les règles d'affaires externalisées du code applicatif. Un moteur de règles constitue un système de production (Cirstea et Kirchner, 2004) au cœur des systèmes experts. Dans le cadre de ce mémoire, l'expertise du système portera sur la prise de décisions dans le domaine de la finance, et plus spécifiquement son secteur social et solidaire. Le terme moteur de règles est assez ambigu car il pourrait qualifier tout système appliquant des règles d'affaires à des données pour produire des conséquences. Des systèmes simples tel que

la validation de formulaires, les nœuds de décision dans un flux de travail et les moteurs d'expressions dynamiques satisfont cette définition. Un moteur de règles au sens *système de production de règles* est une sorte de système expert qui représente les connaissances sous une forme concise, non ambiguë et avec une approche déclarative. Les systèmes simples cités précédemment ne rentrent pas dans cette catégorie.

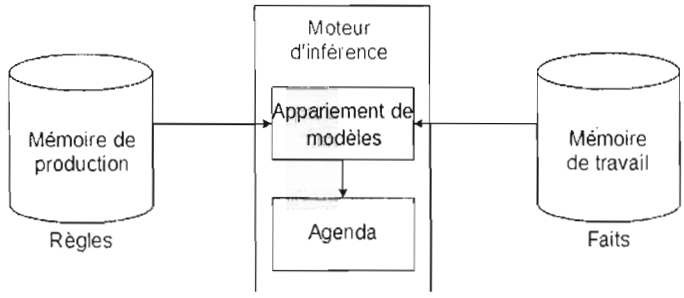


Figure 2.1 Composants d'un moteur de règles

Le moteur de règles doit disposer :

- des types de **faits** : ce sont des entités qui possèdent des attributs. Les types de faits permettent d'organiser les données à manipuler. Par exemple, un type de fait client possèderait les propriétés nom, age, revenu annuel et montant dette. Un client s'appelant Jacques, âgé de 34 ans, ayant un revenu annuel de 70 000 $ et une dette de 25 000 $, constituerait un *fait*.
- de **règles** de production, généralement de la forme SI condition ALORS action. La condition et l'action sont respectivement appelées *Left Hand Side* (LHS) et *Right Hand Side* (RHS). Lorsque le LHS de la règle de production est vérifié, la règle est dite *activée*.

Condition (LHS)	Action (RHS)
Lorsqu'un client demande un prêt de montant x1, qu'il n'a pas de dettes et que son revenu est supérieur à x2	Alors lui accorder le prêt.

Tableau 2.1: Exemple de règle d'affaire

- d'une **mémoire de production**, appelée aussi **base de connaissances** relatives au domaine (KB ou Knowledge Base en anglais). Les connaissances, ou règles de production, sont représentées selon un formalisme interprétable par un système informatique. Les principaux formalismes de représentation des règles standards sont explicités à la section *2.1.4 Standards de représentation des règles d'affaires.*
- d'une **mémoire de travail** ou *working memory* indiquant l'état actuel des faits dans le système. Chaque enregistrement dans cette mémoire est appelé *Working Memory Element (WME).*
- d'un **moteur d'inférence** constituant le cœur du système de production de règles. Il est capable de traiter un large nombre de faits et de règles pour arriver à des conclusions, provoquant elles-mêmes de nouvelles actions. La correspondance entre faits et règles se fait par le moteur d'inférence selon un algorithme d'appariement de modèles tel que Rete, Treat ou Leaps. Lorsque toutes les conditions d'une règle sont vraies pour des faits de la mémoire de travail, la règle est activée et placée dans l'agenda. L'agenda est une collection d'activations (règles éligibles à être exécutées).

Le moteur d'inférence

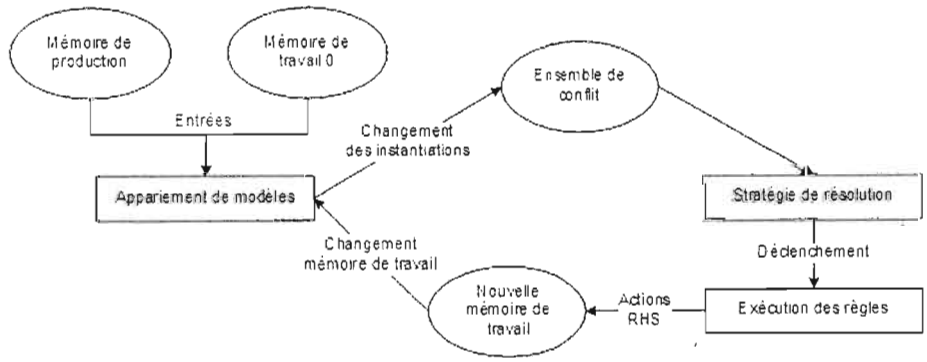


Figure 2.2 Flux de travail d'un moteur d'inférence [d'après (Cirstea et Kirchner, 2004)]

Le moteur d'inférence, sorte de "cerveau" du moteur de règles, passe à chaque itération par trois étapes :

- Les conditions sont vérifiées pour les faits présents dans la mémoire de travail. Chaque

correspondance constitue une *instanciation* de la règle. Elle est formée par une paire {règle, sous-ensemble de données vérifiant les conditions de la règle}. L'ensemble des instanciations forme l'*ensemble des conflits* (conflict set). Les algorithmes d'appariement de modèles (pattern matching), tel que Rete, permettent de rapidement définir ces ensembles de conflits (voir section *2.1.3 Appariement de modèles*).

- Les règles à exécuter à un instant donné sont choisies parmi les règles activées (présentes dans l'agenda) selon une stratégie de sélection. Par exemple, la stratégie LEX donne une plus grande priorité aux règles récemment activées et effectue un tri supplémentaire selon la complexité des conditions des règles (Cirstea et Kirchner, 2004).
- Les règles sélectionnées sont exécutées en leur passant les faits (données) des instanciations comme paramètres. Ceux qui vérifient les conditions des règles exécutées sont alors ajoutés à la mémoire de travail.

Chaque cycle produit de nouvelles conclusions qui pourraient être considérées, à l'itération suivante, comme des faits sur lesquelles s'appliquent des règles. Le processus de déduction peut se répéter pendant un certain nombre de cycles ou jusqu'à ce que plus aucune donnée ne vérifie les règles du système de production.

Stratégies du moteur d'inférence

- Le **chaînage avant** part d'une donnée x et cherche ses conséquences jusqu'à arriver à un but final. Cette stratégie est opportune lorsque la collecte des données est coûteuse, mais qu'elles ne sont pas nombreuses.

  Il existe deux sous-classes de moteurs de règles qui utilisent généralement le raisonnement en chaînage avant :

  - Moteur de règles productif : s'exécute lorsqu'un utilisateur ou une application l'invoque. Par exemple, on répond à la question « Peut-on allouer un prêt au client x ? », en exécutant des règles de traitement des demandes de prêts sur les faits du client x.
  - Moteur de règles réactif : réagit automatiquement quand des événements se pro-

duisent (Événement - Condition - Action). Par exemple, un moteur de règles réactif alerte un gestionnaire lorsque certains produits sont en rupture de stock.

La plupart des moteurs de règles sont à la fois productifs et réactifs, bien qu'ils puissent accentuer la gestion d'un aspect plus que l'autre.

- Le **chaînage arrière** suit le sens contraire du chaînage avant. Il part du but à atteindre et cherche parmi les règles existantes celles qui amènent à ce but. Le chaînage arrière est utile lorsque l'unique but à atteindre est de prouver une conclusion et que les valeurs des autres faits ne sont pas intéressantes. Son utilisation est pertinente lorsque la quantité de données est potentiellement très large et que l'intérêt se limite à certaines caractéristiques spécifiques du système. Typiquement, le chaînage arrière est utilisé dans les problèmes de diagnostic, tel que le diagnostic médical ou la localisation d'erreurs dans des équipements mécaniques. La plupart des systèmes experts de première génération étaient basées sur une forme de chaînage arrière, à l'exception de quelques langages de production de règles, tel que OPS5, qui utilisaient le chaînage avant.
- Une **stratégie mixte** consiste à utiliser les deux formes d'inférence. En pratique, la plupart des raisonnements sont un mélange du chaînage avant et du chaînage arrière : Étant donné une supposition initiale, on infère une conclusion en raisonnant "vers l'avant". On applique ensuite un chaînage arrière pour trouver d'autres données qui confirment les conclusions.

  On peut aussi faire du chaînage arrière vers les causes plausibles d'un but puis faire du chaînage avant pour exploiter les conséquences des nouvelles données. Cette approche appelée "chaînage opportuniste" car les conséquences des données sont exploitées au fur et à mesure qu'elles deviennent disponibles.

  Le chaînage mixte est implémenté dans les moteurs de règles les plus perfectionnés.

2.1.3 Appariement de modèles (Pattern Matching)

Certains moteurs de règles basiques ne font que chaîner la logique procédurale dans un ordre spécifié par l'utilisateur. La plupart des moteurs de règles évolués utilisent des

algorithmes d'appariement sophistiqués tel que Rete, Treat et Leaps. L'appariement de modèles consiste à trouver toutes les règles de la mémoire de production dont toutes les conditions peuvent être vérifiées par une combinaison de faits de la mémoire de travail. Les moteurs d'inférence déterminent les règles à exécuter et leur ordre d'exécution. Ceci constitue une amélioration par rapport au paradigme procédural, où un unique programme exécute une suite d'instructions une seule fois.
À titre d'illustration, on considère une mémoire de travail dont les éléments sont représentés par des triplets (identifiant, attribut, valeur).
Les éléments sont :

w1 : (CL1 client-de BQ1)	w4 : (BQ1 nature sociale)
w2 : (CL2 client-de BQ2)	w5 : (BQ2 nature traditionnelle)
w3 : (CL3 client-de BQ2)	w6 : (BQ3 nature sociale)

Soit la règle p1 suivante qui s'applique aux clients de banques de nature sociale :

```
p1: (<x1> client-de <y1>) (<y1> nature <sociale>)
        -> actions a executer)
```

La production p1 est vérifiable pour la mémoire de travail décrite précédemment car les trois conditions sont satisfaites par les WMEs w1 et w4. Les variables x1 et y1 prennent respectivement les valeurs CL1 et BQ1. Ainsi, p1 fera partie de *l'ensemble des conflits* de règles éligibles à être exécutées par le moteur de règles et sera placée dans l'agenda.

Algorithme Rete

Rete est un algorithme d'appariement très utilisé par les moteurs de règles[1]. Le mot Rete signifie réseau en italien. Les réseaux Rete sont composés d'une partie alpha et d'une partie beta.

- La partie alpha contient une mémoire alpha pour chaque symbole constant utilisé dans les conditions. Une mémoire alpha contient l'ensemble des WMEs qui répondent

1. Clips, Drools de JBoss, JRules, Jess utilisent différentes versions de Rete

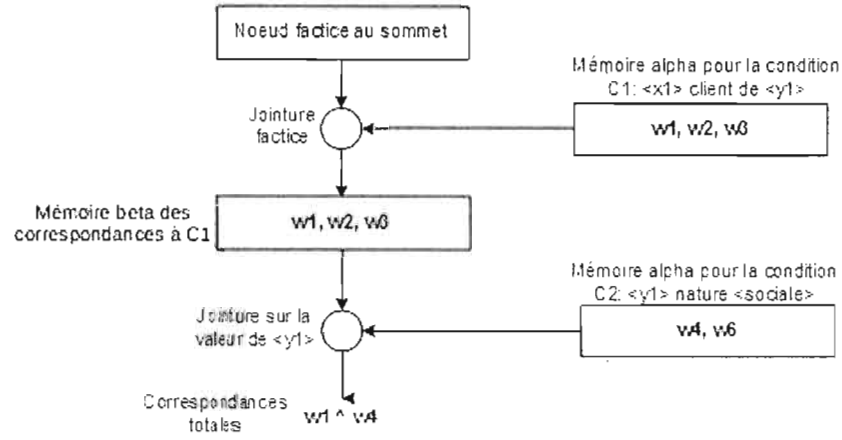


Figure 2.3 Illustration de l'appariement de modèles de Rete

à la condition représentée par la mémoire. Pour l'exemple précédent, la mémoire alpha pour la condition *<x1> client-de <y1>* contient les éléments w1, w2 et w3. De même, celle de la condition *<y1> nature <sociale>* contient les éléments w4 et w6.

- La partie beta contient des jointures entre des ensembles d'éléments. Le résultat d'une jointure est stocké dans une mémoire beta. Par exemple, une mémoire beta pourrait stocker les éléments qui répondent à la fois à la condition *<x1> client-de <y1>* et *<x2> nature <sociale>*.

Ainsi, le réseau alpha effectue des tests pour des éléments individuels tandis que le réseau beta effectue des tests sur des combinaisons de WMEs. Le réseau ainsi obtenu forme une sorte de machine à états. Les faits entrent dans l'arbre aux nœuds du plus haut niveau et descendent dans l'arbre s'ils répondent aux conditions, jusqu'à ce qu'ils atteignent les feuilles, qui représentent les conséquences des règles. La figure 2.3 illustre le fonctionnement du réseau Rete pour notre exemple.

Charles Forgy, l'auteur de Rete, et d'autres ont tenté d'améliorer l'algorithme en donnant naissance à Treat (Wright et Marshall, 2003), Leaps (Batory, 1994), Rete II et Rete III.

Les algorithmes de matching tel que Rete sont plus complexes que le simple fait d'exécuter une série de bouts de code dans l'ordre. Si l'on exécute chaque règle sur chaque fait dans

l'ordre, un moteur de règles sera alors un facteur de ralentissement et non d'amélioration des performances. La sophistication de l'algorithme de matching permet de gagner du temps grâce à des mécanismes tel que le partage de conditions, où une condition partagée par plusieurs règles peut être évaluée une seule fois pour toutes les règles.

2.1.4 Standards de représentation des règles d'affaires

Cette section explique la spécification JSR-94 et la représentation de règles d'affaires sous forme de tables et d'arbres décisionnels. Il n'existe jusqu'à présent aucun format de représentation des règles qui soit une norme *de facto*. Cependant, certaines initiatives tel que RuleML[2], RIF[3] (Rule Interchange Format) et SBVR[4] (Semantics of Business Vocabulary and Business Rules) augurent d'une meilleure interopérabilité dans le futur.

JSR-94[5]

JSR-94 est une spécification qui définit une API simple permettant d'accéder aux moteurs de règles à partir de clients Java.
La plupart des moteurs de règles possèdent des APIs propriétaires, ce qui les rend difficiles à intégrer aux applications. Si un organisme change de moteur de règles, la plupart du code applicatif interagissant avec le moteur devra être réécrit. JSR-94 tente de remédier à ce problème en standardisant les implémentations de moteurs de règles pour Java.
JSR-94 fournit des lignes directrices pour les APIs d'administration et d'exécution des règles. Elle encapsule l'ajout et la suppression de règles, leur analyse grammaticale, l'inspection de leurs métadonnées, leur exécution ainsi que la récupération et le filtrage des

2. http ://ruleml.org/ (dernière consultation le 09-11-10)

3. http ://www.w3.org/2005/rules/wiki/RIF_Working_Group (dernière consultation le 09-11-10)

4. http ://www.businessrulesgroup.org/sbvr.shtml (dernière consultation le 09-11-10)

5. http ://jcp.org/en/jsr/detail ?id=94 (dernière consultation le 09-11-10)

résultats d'exécution. En revanche, la spécification ne standardise pas le fonctionnement du moteur de règles, ni le flux d'exécution des règles, ni leur langage de description. Les APIs de JSR-94 sont séparées en deux catégories [6] :

- API d'administration des règles : fournit les classes de chargement des règles et des actions qui leur sont associées en tant qu'ensembles d'exécution. Les règles peuvent être chargées à partir de ressources externes tel que des URIs, des fichiers XML ou encore des arbres binaires. L'API fournit aussi les méthodes pour ajouter et supprimer des ensembles d'exécution de règles. Elle permet également de définir des permissions sur ces ensembles afin de restreindre les autorisations d'accès.
- API d'exécution client : fournit les classes utilisées par les clients pour exécuter les règles et récupérer les résultats. Seules les règles enregistrées via l'API d'administration sont accessibles. L'API permet aussi aux clients de définir des sessions de règles et d'exécuter les règles à l'intérieur de ces sessions.

Tables et arbres décisionnels

Les tables et les arbres décisionnels sont des formats couramment utilisés dans les entreprises pour exprimer une logique métier. Plusieurs systèmes de gestion de règles sont capables d'importer des définitions à partir de tables et arbres décisionnels.

- Tables décisionnelles : Elles constituent un moyen précis et compact de modéliser une logique complexe. À l'instar des constructions *if..else* et *switch..case*, les tables décisionnelles associent des conditions à des actions. Elles permettent d'associer plusieurs conditions indépendantes avec plusieurs actions d'une manière élégante. Elle sont également plus faciles à lire et à examiner que du code utilisant des structures de contrôle et sont souvent utilisées en entreprise pour produire les spécifications.
- Arbres de décision : Un arbre de décision représente graphiquement les règles selon une arborescence. Les nœuds représentent des conditions, leurs ramifications représentent

---

6. http ://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html (dernière consultation le 09-11-10)

SI le client a une hypothèque	ET sa note de crédit externe est	ET son solde carte crédit est	ALORS la recommandation est
Oui			Haute
Non	[100 – 550]		Haute
Non	[550 – 900]	<=0	Moyenne
Non	[550 – 900]	>0	Haute

Figure 2.4 Table décisionnelle

les différents résultats d'évaluation de la condition. Avec l'introduction des probabilités, les arbres de décision deviennent des outils puissants car ils permettent d'évaluer des chemins complets en combinant les probabilités de l'ensemble des nœuds formant chaque séquence de décisions. Ainsi, une décision pourrait être individuellement moins judicieuse que d'autres, mais en explorant plus en profondeur l'arbre de décisions, elle pourrait s'avérer meilleure. Cette application de la théorie des probabilités est utile pour résoudre un large éventail de problèmes décisionnels. Il est également possible d'utiliser des facteurs de certitude au lieu des probabilités. Toutefois, les experts métier ne visualisent habituellement pas les connaissances à la manière d'un arbre de décision.

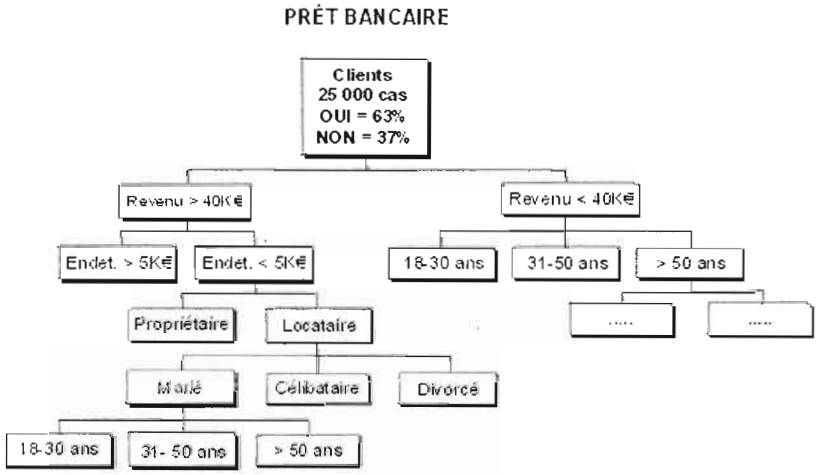


Figure 2.5 Arbre de décision[7]

2.1.5 Différence entre moteur de règles et système de gestion de règles

Un moteur de règles d'affaires est une partie d'un système global qui s'occupe de tout ce qui est associé à l'utilisation des règles. Bien qu'il constitue une partie essentielle, son rôle se limite à l'exécution : quelles règles doivent être exécutées et dans quel ordre ? Dans le cadre d'une utilisation en entreprise, incluant le développement, les tests, la liaison vers des applications externes et le déploiement dans des environnements divers ; les outils en addition au moteur sont critiques dans la gestion des règles d'affaires.
En disposant d'interfaces accessibles et d'une représentation des informations selon les termes métier, les systèmes de gestion de règles d'affaires sont capables de maintenir les stratégies d'affaires à travers les systèmes d'information de l'entreprise. Ceci leur permet de modifier le comportement des applications sans l'assistance du personnel informatique, qui peut se concentrer sur des tâches à plus grande valeur ajoutée.
Les systèmes de gestion de règles requièrent un dépôt de données, des applications de maintenance de règles, des outils de vérification aussi bien que le moteur de règles lui-même. Les produits les plus importants du secteur se définissent comme des systèmes de gestion de règles et non pas comme de simples moteurs de règles.
Notre évaluation englobera le système complet et ne se limitera pas au moteur de règles. Tout au long du mémoire, on parlera de "moteur de règles" pour indiquer le moteur d'exécution et de "système de gestion de règles" pour indiquer le système composé du moteur et des outils de conception et de maintenance des règles. Ces outils sont en effet d'une importance capitale dans les environnements où sera déployée la solution à recommander pour la chaire.

---

7. http ://www.aiaccess.net/French/Glossaires/GlosMod/images/ArbredeDecision1.gif (dernière consultation le 09-11-10)

2.1.6 Apports des moteurs de règles

Permettre le changement rapide des règles

Dans le cas d'un système informatique libre de finance sociale et solidaire, l'utilité d'un moteur de règles d'affaires est d'autant plus réelle que le système libre est destiné à être assez générique, car il sera réutilisé :

- dans plusieurs secteurs qui possèdent chacun leurs propres règles d'affaires (assurances, placements en bourses...)
- pour un même secteur, avec plusieurs variantes (selon le pays, selon les règles internes des organismes...)

Par ailleurs, les règles d'affaires au sein d'un même organisme sont variables et doivent être découplées du code applicatif. Le fait de les réunir à un endroit unique facilite leur maintenance et leur modification.

Donner le pouvoir aux utilisateurs

La philosophie de la Chaire est de déléguer aux utilisateurs la paramétrisation de leurs propres systèmes selon leurs besoins, sans qu'ils aient à recourir aux services de tierces parties, et ce dans la mesure du possible. Plusieurs moteurs de règles d'affaires existants permettent d'utiliser un langage proche du langage naturel. Ainsi, les règles sont compréhensibles par un non informaticien. Leur maintenance et leur modification peut s'effectuer sans avoir recours aux services, coûteux et parfois non disponibles, de tierces parties.

Améliorer la lisibilité des règles d'affaires

L'évaluation en code applicatif traditionnel des combinaisons d'une série d'instructions conditionnelles génère l'écriture de logique profondément imbriquée pour résoudre un problème. Le fait de définir les règles dans un style déclaratif et non impératif fait qu'on définit individuellement les règles. On obtient ainsi une représentation plus lisible,

débarrassée des imbrications de conditions que rajouterait le style impératif d'un langage comme Java. D'autre part, le style déclaratif utilisé pour les règles définit quoi faire et non comment le faire. Ceci aboutit à produire une représentation consistante des règles en supprimant les éventuelles variations de codage du "comment faire". La consistance de représentation permet de facilement étendre les règles et d'y greffer des fonctionnalités tel que l'audit, la journalisation et l'optimisation de performances.

Améliorer les performances

Les algorithmes d'appariement de modèles et le partage de conditions permettent un traitement plus performant qu'avec l'utilisation des conditions imbriquées d'un langage impératif comme Java. De plus, les moteurs de règles supportent un volume conséquent de règles d'affaires, et sont donc aptes à faire face à l'évolution de la complexité des systèmes.

2.1.7 Quand opter pour un moteur de règles ?

L'adoption d'un moteur de règles peut susciter des réticences dans certains organismes. La valeur ajoutée ne leur semble pas justifier l'effort d'apprentissage et de déploiement de cette technologie pas encore largement adoptée.
Cette section présente une liste de vérification, inspirée des directives du professeur George Rudolph [8]. Un organisme ayant des doutes sur la pertinence de l'utilisation d'un moteur de règles devrait s'assurer de passer les points de la liste en les parcourant du premier au dernier, dans l'ordre.

Nature des algorithmes

Les algorithmes de l'organisme relèvent-ils principalement du calcul intensif ou impliquent-ils un volume conséquent de prises de décisions.

---

8. http ://herzberg.ca.sandia.gov/guidelines.shtml (dernière consultation le 09-11-10)

- Si les algorithmes utilisés sont principalement de calcul intensif, sans beaucoup de prises de décisions, alors l'utilisation d'un moteur de règles n'est pas recommandée.
- Si les algorithmes utilisés comprennent plusieurs ramifications conditionnelles et prises de décisions ; si le code applicatif contient beaucoup de *if*, *else*, *switch* et une logique confuse qui doit être constamment réarrangée, alors l'utilisation d'un moteur de règles est à considérer.

Complexité des décisions

Les décisions qui doivent être faites sont-elles relativement simples ou complexes ?

- Si l'organisme n'arrive pas à formaliser les règles d'affaires, pour quelque raison que ce soit, il faudrait alors se pencher sur ce problème avant d'envisager l'utilisation d'un moteur de règles.
- Si les règles contiennent en moyenne 2 conditions ou moins, l'utilisation de moteurs de règles serait exagérée.
- Si les règles contiennent en moyenne 3 conditions ou plus, l'utilisation d'un moteur de règles est à considérer.

Fréquence de changement des règles

Les règles sont-elles statiques ou varient-elles dans le temps ?

- Si la logique est bien définie et statique, la flexibilité qu'apportent les moteurs de règles n'est pas pertinente.
- Si les règles sont susceptibles de changer dans le temps, en vue de la nature de l'application, alors le coût de l'utilisation d'un moteur de règles est compensé par la flexibilité apportée.

Disponibilité de ressources

L'organisme peut-il supporter les coûts totaux de l'utilisation d'un moteur de règles pendant le cycle de vie entier du système ?

- Si le calendrier de l'organisme ne peut s'accommoder du déploiement d'un système de gestion de règles, alors son utilisation n'est pas recommandée.
- Si l'organisme ne peut attendre un an après le déploiement du système de gestion de règles, pour commencer à observer les gains dûs à la flexibilité et à la facilité d'utilisation apportés, alors son utilisation n'est pas recommandée.
- Si l'organisme n'est pas prêt à former ses développeurs et ses experts métiers, l'utilisation d'un système de gestion de règles n'est pas recommandée.
- Si les ressources de l'organisme sont disponibles, l'utilisation d'un système de gestion de règles vaut largement l'investissement.

2.2 Les moteurs de flux de travail

Les moteurs de flux de travail s'inscrivent dans le contexte des processus d'affaires. Afin de comprendre leur rôle, il convient de définir les notions élémentaires de *flux de travail*, *processus d'affaires* et autres *activités*, *participants* et *items de travail.* La figure 2.6 schématise les relations entre ces éléments.

2.2.1 Processus d'affaires (Business Process)

(Davenport, 1993) définit les processus d'affaires comme :

> « Un ensemble structuré, mesuré d'activités conçues pour produire une sortie spécifique pour un client ou un marché particulier. Il implique [le processus d'affaires] de mettre l'accent sur *la manière* dont le travail est fait dans une organisation[...]. Un processus est ainsi un ordonnancement spécifique d'activités de travail à travers le temps et l'espace, avec un début et une fin, ainsi que des entrées et des sorties clairement définies. »

Un processus d'affaires n'a donc de sens que dans le cadre d'un organisme. Son exécution peut être déclenchée automatiquement lorsque certaines conditions sont vérifiées.

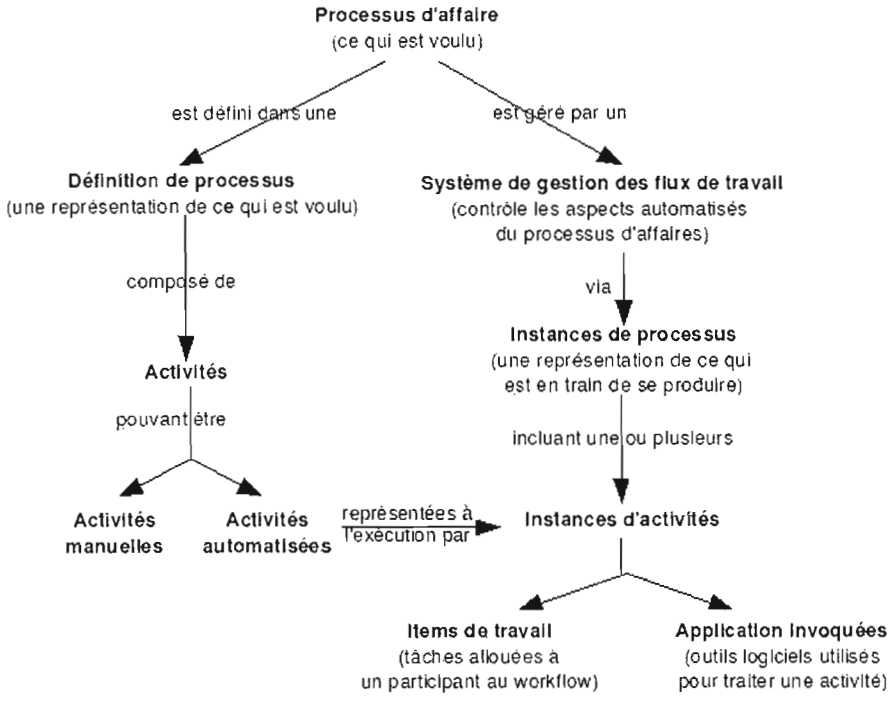


Figure 2.6 Terminologie des flux de travail (Coalition, 1999)

Activités

Un processus d'affaires est formé par la combinaison logique de plusieurs activités. L'activité constitue la plus petite unité de travail pouvant être planifiée par un moteur de flux de travail. Les activités peuvent être automatiques ou manuelles. Une activité automatique est assignée à un participant au processus d'affaires. Elle peut invoquer l'utilisation d'outils et d'applications tierces dans le cadre de son exécution. Une activité manuelle est réalisable par un participant humain.

Instances de processus d'affaires

Le moteur de flux de travail peut lancer une ou plusieurs instances d'un même processus d'affaire. La définition d'un processus établit ses activités et procédures. Les instances créées à partir de cette définition possèdent chacune leur propre état et leurs propres

données locales.

2.2.2 Flux de travail

Les flux de travail ou workflow permettent la modélisation et la gestion informatique de l'ensemble des processus métier à accomplir et des différents participants impliqués dans leur réalisation. En effet, un flux de travail décrit le circuit de validation, les tâches à accomplir entre les différents participants au processus, les délais et les modes de validation. Il fournit à chacun des participants les informations nécessaires pour la réalisation de ses tâches. Il permet généralement un suivi de l'état d'avancement du flux.
La WfMC (Coalition, 1999) définit le flux de travail comme :

> « L'automatisation partielle ou totale des processus d'affaires, pendant laquelle documents, informations et tâches sont passés d'un participant à un autre pour action, selon un ensemble de règles procédurales. » [Notre traduction]

2.2.3 Définition des moteurs de flux de travail

Un moteur de flux de travail est un service logiciel fournissant l'environnement d'exécution pour des instances de processus.
Il permet :

- L'interprétation des définitions de processus.
- La création d'instances de processus et la gestion de leur exécution (démarrage, arrêt, suspension..).
- La navigation entre activités et la création d'unités de travail appropriées pour leur traitement.
- La supervision et la gestion des flux de travail.

Participants et listes de travail

Un participant dans un flux de travail accomplit les tâches de sa liste de travail (*work list*). Les unités de travail (*work items*) qu'elle contient servent à l'accomplissement des activités des instances de processus d'affaires actives dans le moteur de flux de travail. La liste de travail est créée soit à l'initiative du participant via une requête vers le moteur de travail, soit à l'initiative du moteur de travail lui même. Ce dernier va suivre le cheminement décrit par les processus instanciés et présenter aux participants concernés la/les activités à réaliser.

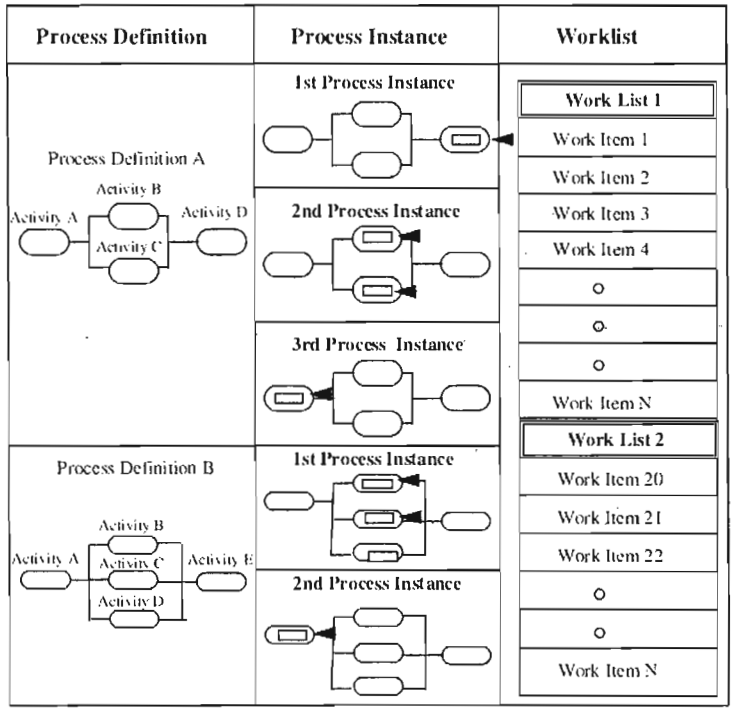


Figure 2.7 Listes et unités de travail (Coalition, 1999)

Modèle de référence du flux de travail

La WfMC (Workflow Management Coalition) a développé une représentation architecturale d'un système de gestion de flux. Ce modèle de référence identifie cinq interfaces

de communication entre un moteur de flux de travail et son environnement. L'interface 1 concerne l'importation et l'exportation de définitions de processus. L'interface 2 représente les interactions entre les applications client et le module de gestion des listes de travail. L'interface 3 englobe les invocations d'outils logiciels et d'applications de tierce partie. Enfin, les interfaces 4 et 5 représentent respectivement la communication avec d'autres moteurs de flux de travail et l'utilisation d'outils d'administration et de suivi des activités.

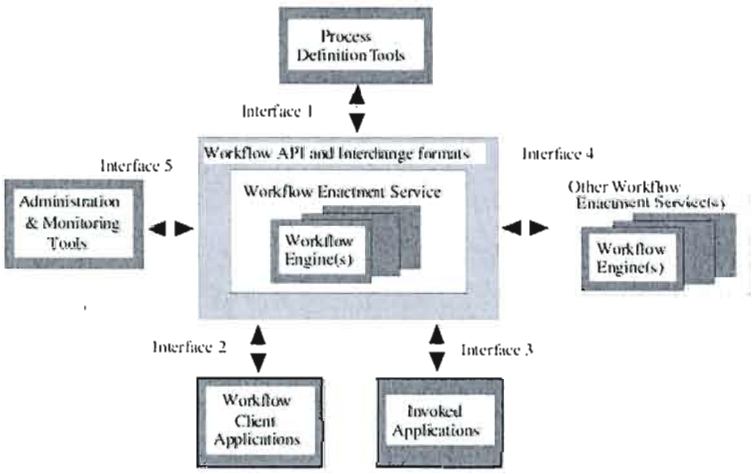


Figure 2.8 Modèle de référence du flux de travail (Coalition, 1999)

2.2.4 Langages de définition des processus d'affaires

À leurs débuts, les moteurs de flux de travail proposaient chacun leur propre langage de définition des processus d'affaires. Il pouvait s'agir de langages de programmation classiques ou de formats créés spécifiquement pour le moteur. Le besoin d'échanger les flux de travail s'est rapidement fait sentir. C'est alors que des standards ont été établis afin de fournir une modélisation et un langage de définition communs. La notation BPMN (Business Process Modeling Notation) s'impose comme la norme *de facto* de la modélisation graphique des processus d'affaires. En quelque sorte, BPMN est à la modélisation des flux de travail ce que UML est à la modélisation des systèmes infor-

matiques orientés objets.

Pour être exécutés par les moteurs de flux de travail, ces flux doivent être implémentés dans un langage. L'organisme OASIS (Organization for the Advancement of Structured Information Standards) [9] a défini le langage BPEL (Business Process Execution Language) qui permet d'exécuter des instances de processus. De son côté, le consortium WfMC (Workflow Management Coalition) [10] a spécifié un format d'échange commun de définitions de processus d'affaires nommé XPDL (XML Process Definition Language). Certains moteurs de flux de travail libres, tel qu'Apache ODE, exécutent des définitions de processus écrites au format BPEL. D'autres, comme Enhydra Shark et WfMOpen se basent sur la spécification XPDL. Plusieurs de ces moteurs rajoutent leurs propres extensions afin d'apporter des fonctionnalités supplémentaires. Il existe encore des moteurs de flux de travail qui utilisent leurs propres langages de définition des processus ; à l'instar de *Ruote* qui utilise un DSL (Domain Specific Language) Ruby et de JBoss jBPM qui a créé le langage jPDL (Java Process Definition Language).

Les sous-sections suivantes traitent de la norme de modélisation graphique BPMN, du langage de définition XPDL et du langage d'exécution BPEL.

Business Process Modeling Notation (BPMN)

BPMN est une notation standard pour la modélisation des processus d'affaires. Elle fût développée par la *Business Process Management Initiative* et est actuellement prise en charge par l'*Object Management Group*. BPMN définit les diagrammes de processus d'affaires BPD (Business Process Diagram) qui représentent graphiquement des activités et les contrôles de flux qui définissent leur ordre d'exécution. La norme vise à fournir une notation compréhensible à tous les niveaux, des analystes métier aux utilisateurs en passant par les informaticiens. Les diagrammes BPD sont faciles à utiliser et à comprendre tout en étant expressifs et capables de représenter des processus d'affaires

9. http ://www.oasis-open.org/ (dernière consultation le 09-11-10)

10. http ://www.wfmc.org/ (dernière consultation le 09-11-10)

complexes (Owen et Raj, 2003).

BPMN peut faire le pont entre les modélisations métier et les langages informatiques. En effet, il est possible de transformer automatiquement des diagrammes BPMN en code exécutable BPEL ou XPDL et vice-versa (voir sections suivantes).

BPMN définit quatre catégories d'éléments graphiques [11] :

- Objets de flux : regroupent les événements, activités et passerelles (gateways). Les événements peuvent être déclencheurs ou résultats. Les activités sont soit des tâches atomiques, soit des sous-processus. Enfin, les passerelles contrôlent la divergence et la convergence des flux séquentiels en flux concurrentiels.
- Connecteurs : ce sont les flux de séquences, les flux de messages et les associations. Ils servent à relier les objets de flux.
- Bassins (pools) et couloirs (swimlanes) : organisent les activités selon leurs responsabilités ou selon leurs capacités fonctionnelles.
- Artefacts : définissent le contexte des processus d'affaires. Il peut s'agir d'annotations textuelles, de représentation de données, de groupes ou encore d'artefacts personnalisés.

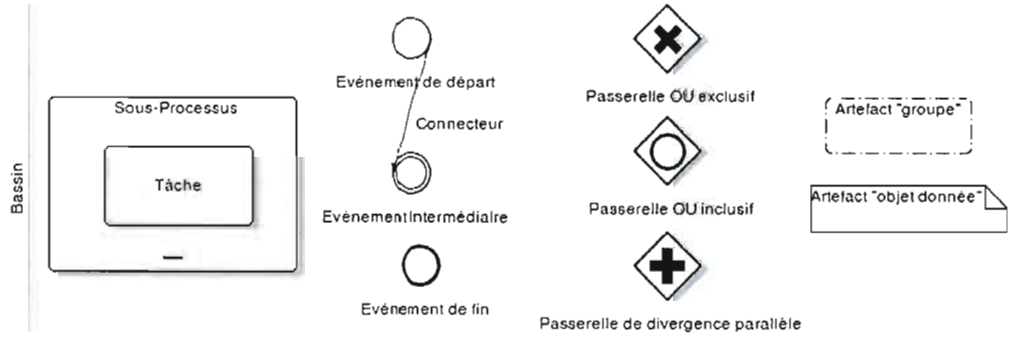


Figure 2.9 Exemples d'éléments graphiques BPMN

La figure 2.10 est une représentation selon la norme BPMN du flux de travail pour un traitement simple d'une demande de prêt. Le diagramme est aisément compréhensible, aussi bien par les experts métier que les experts informatiques.

11. (Pant, 2008)

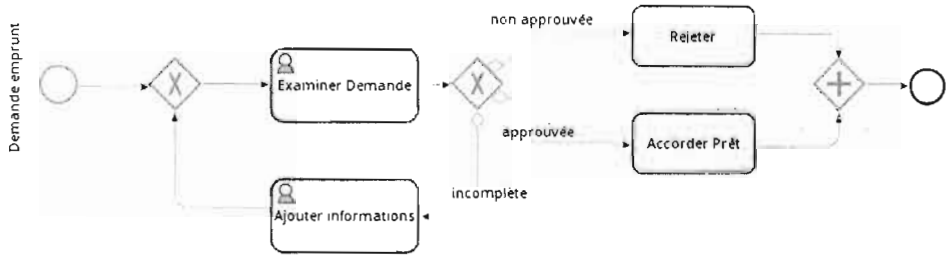


Figure 2.10 Notation BPMN d'une demande de prêt simple

XML Process Definition Language (XPDL)

XPDL est une initiative de la coalition pour la gestion du flux de travail WfMC. WfMC est un consortium formé pour définir les standards d'interopérabilité entre systèmes de gestion de flux de travail. Plus de 300 organismes internationaux dont IBM, Hewlett-Packard et Fujitsu en font partie.

Il s'agit d'un langage basé sur XML qui se revendique comme le format de sérialisation des diagrammes BPMN [12]. Un fichier XPDL forme un paquetage contenant les définitions d'un à plusieurs processus. Les informations pour chaque processus incluent les participants, artefacts et autres objets de données. Il est possible de définir les informations sur la représentation graphique des éléments du processus.

Le langage permet d'invoquer des applications et des outils requis par les processus sous forme d'objets simples Java (Plain Old Java Object), d'EJBs (Enterprise Java Beans), de scripts, de services web ou encore de règles d'affaires.

XPDL sert de format d'échange commun des définitions de processus à travers les différents outils de gestion de flux (Coalition, 2005). Il permet de séparer la définition des processus de leur exécution. Ainsi, un processus défini en XPDL peut être exécuté sur différents moteurs de flux de travail. Chaque outil implémentant XPDL peut rajouter des attributs propriétaires. Si un outil ne comprend pas certaines extensions, il traitera la partie standard sans éditer la partie propriétaire qui sera ainsi préservée. Le fait de

12. http ://www.xpdl.org/ (dernière consultation le 09-11-10)

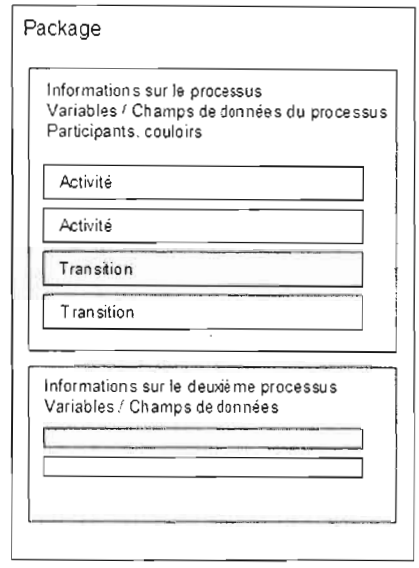


Figure 2.11 Structure d'un fichier XPDL

définir un ensemble d'éléments communs à toutes les implémentations XPDL, tout en permettant d'ajouter des extensions fait de XPDL un format portable et extensible.
Pour justifier sa revendication d'être le format de sérialisation des diagrammes BPMN, XPDL formalise le lien d'équivalence entre les éléments de la notation BPMN et les structures XPDL qui y correspondent ; et ce dans la spécification même du standard (Coalition, 2008). Les blocs *<xpdl :Activity>* définissant des activités correspondent aux nœuds BPMN (passerelles, activités et événements) ; les structures de type *<xpdl :Transitions>* représentent des connexions...
(White, 2003) illustre les transformations BPMN-XPDL en prenant des diagrammes BPMN et en implémentant en code XPDL leurs éléments un à un. La figure 2.12 en est une illustration. Le but d'une telle approche est d'amener les experts métier utilisant la notation BPMN à utiliser XPDL pour l'implémentation de leurs processus d'affaires.
On clôture cette section avec des blocs de code illustratifs du langage XPDL.

Listing 2.1 Représentation des activités en XPDL

```
<xpdl:Activity Id="7ADB5EA3F8" Name="Decide on Claim">
<xpdl:Object Id="1D-9BCB-707ADB5EA3F8" />
```

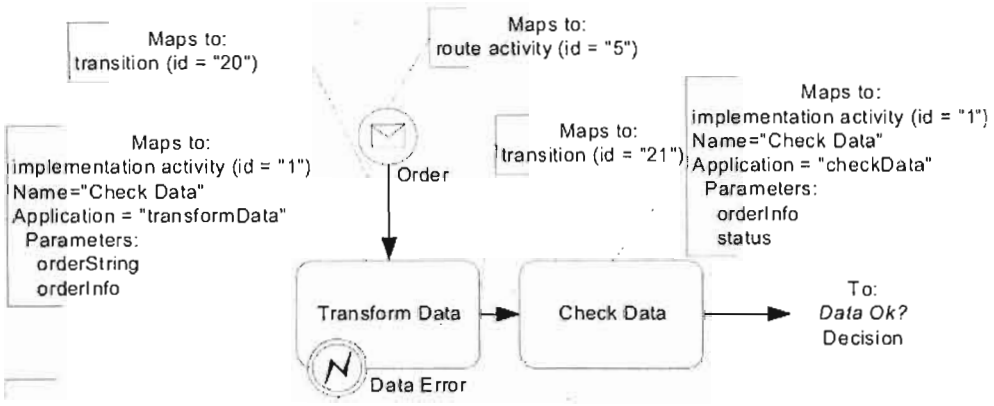


Figure 2.12 Implémentation d'un diagramme BPMN en XPDL (White, 2003)

Listing 2.2 Contrôle de flux avec un OU exclusif

```
<xpdl:Route GatewayType="XOR" />
<xpdl:TransitionRestrictions>
  <xpdl:TransitionRestriction>
    <xpdl:Split Type="XOR" />
```

Listing 2.3 Stockage des coordonnées pour la représentation graphique

```
<xpdl:NodeGraphicsInfos>
  <xpdl:NodeGraphicsInfo Width="129" Height="80" LaneId="1">
    <xpdl:Coordinates XCoordinate="100" YCoordinate="430" />
  </xpdl:NodeGraphicsInfo>
</xpdl:NodeGraphicsInfos>
```

Parallèlement à l'initiative XPDL, un langage nommé BPEL tente de s'imposer comme le standard d'orchestration des flux de travail.

Business Process Execution Language (BPEL)

Le standard BPEL est une initiative de l'Organisation pour l'Avancement des Standards d'Information Structurée. OASIS est un consortium à but non lucratif dirigé par des acteurs majeurs de l'industrie informatique dont Oracle, IBM, SAP et Microsoft. Il se

fixe comme mission la « conduite du développement, de la convergence et de l'adoption de standards ouverts pour la société d'information ». Dans ce cadre, l'organisme produit des standards pour les services web, la sécurité et le commerce électronique.

BPEL est un langage basé sur XML permettant de définir des processus d'affaires qui seront exécutés par un moteur d'orchestration. L'orchestration peut se faire entre des entités internes à l'entreprise ainsi qu'avec des entités externes. Les informations sont exclusivement échangées par le biais d'interfaces de services web. Les processus peuvent envoyer et recevoir des messages SOAP (Simple Object Access Protocol). Il est possible avec BPEL d'invoquer plusieurs services web simultanément et de synchroniser les résultats.

Les services web "classiques" définis avec le langage WSDL (Web Service Definition Language) ont un modèle d'interaction sans sauvegarde d'état. BPEL étend WSDL et rajoute la possibilité d'exprimer des interactions avec sauvegarde d'état grâce notamment au type *<bpel :variables>* de BPEL.

Le langage ne gère pas la représentation graphique des processus. D'autre part, BPEL est un langage de haut niveau qui permet aux experts métier d'implémenter les flux logiques qui combinent des fonctions afin de résoudre des problèmes plus complexes. L'implémentation des fonctions de bas niveau est réalisée avec des langages de programmation comme Java (Blow et al., 2004).

La définition d'un processus en BPEL se fait via des éléments spécifiés dans le langage dont :

- imports : des imports de fichiers WSDL qui définissent des interactions ou de fichiers XML qui définissent des types de données utilisés dans le processus.
- extensions : l'éventuelle déclaration d'un espace de noms contenant des extensions du langage BPEL par de nouveaux attributs et éléments.
- partnerlinks : décrivent les interactions avec les partenaires d'affaire. La liste des services participants au processus BPEL est encapsulée dans une balise *<bpel :partnerLinks>*.
- échanges de messages : corrèlent les activités entrantes avec les réponses associées.
- variables : données stockant l'état actuel du processus d'affaires ou données échangées avec les partenaires. La liste des messages et des documents XML utilisés dans le

processus BPEL est encapsulée dans la balise *<bpel :variables>*. La variable *input* est une référence au message passé en entrée pendant l'initialisation. La variable *output* est une référence au message qui sera envoyé en réponse au demandeur.

- ensembles de corrélation : ensemble de champs de données de l'application qui permet d'identifier une conversation.
- gestionnaires d'événements et gestionnaires d'erreurs.
- activité primaire : exécute la logique du processus. Définie par un flux d'activités primaires (invoquer, assigner, attendre...)

Le bout de code suivant montre une partie de la définition d'une demande de prêt avec BPEL. La balise d'en-tête indique le nom du processus et le fichier de définition du format BPEL. Vient ensuite la déclaration du partenaire *client*. Ses nom, type et rôle dans le cadre du processus sont définis. *<bpel :variables>* englobe les variables d'entrée et de sortie du processus qui sont en l'occurrence des messages de demande de prêt et de réponse à la demande. Enfin, la logique d'orchestration définit l'ensemble des activités coordonnant le flux de messages à travers les services intégrés à la demande de prêt. L'attribut *operation* des activités *<bpel :receive>* et *<bpel :invoke>* indique la méthode réalisant la tâche de bas niveau requise par l'activité. BPEL permet d'invoquer du code Java ou encore des opérations encapsulées dans des services web.

Listing 2.4 Définition d'un processus avec BPEL

```
<bpel:process name="demandepret"
        xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
            executable">
    <bpel:partnerLinks>
        <bpel:partnerLink name="client"
                    partnerLinkType="tns:demandepret"
                    myRole="demandepretProvider"
                    partnerRole="demandepretRequester"/>
    </bpel:partnerLinks>
    <bpel:variables>
        <bpel:variable name="input"
                    messageType="tns:demandepretRequestMessage"/>
        <bpel:variable name="output"
```

```
                messageType="tns:demandepretResponseMessage"/>
    </bpel:variables>
    <bpel:sequence name="main">
        <bpel:receive name="receiveInput" partnerLink="client"
                portType="tns:demandepret"
                operation="initiate" variable="input"
                createInstance="yes"/>
        <bpel:invoke name="callbackClient"
                partnerLink="client"
                portType="tns:demandepretCallback"
                operation="onResult"
                inputVariable="output"/>
    </bpel:sequence>
</bpel:process>
```

La transformation des diagrammes BPMN en code BPEL est illustrée par de nombreux exemples dans la spécification de BPMN et dans de nombreuses autres publications. (Gao, 2006) note que, même si cette transformation est souvent possible, elle est intrinsèquement complexe vu la « disparité structurelle » entre BPMN, qui produit des graphiques à format relativement libre, et BPEL, qui définit un format structuré en blocs. Structurellement, BPMN est donc un super-ensemble de BPEL. Ainsi, tout processus déclaré sous format BPEL peut être représenté graphiquement avec la notation BPMN mais l'inverse n'est pas toujours vrai.

2.2.5 Différence entre moteur de flux de travail et système de gestion de flux de travail

À l'instar des moteurs de règles, les moteurs de flux de travail constituent une partie d'un système global qui s'occupe de tout ce qui est associé à la gestion des flux de travail. Bien qu'ils constituent une partie essentielle, leur rôle se limite à l'exécution d'instances de processus.

Les systèmes de gestion de flux de travail offrent des outils facilitant la conception et la compréhension des processus ainsi que la collaboration entre les différents acteurs

impliqués (Software, 2008).

Ces systèmes permettent une modélisation graphique des processus à l'aide de notations tel que BPMN. Ceci facilite la compréhension et l'échange de processus entre informaticiens et experts métier.

De plus, les systèmes les plus perfectionnés reflètent graphiquement et en temps réel l'exécution des instances de processus. Certains systèmes permettent de générer automatiquement les interfaces graphiques d'interaction avec les participants humains.

Par ailleurs, les détails d'implémentation peuvent être abstraits grâce à des assistants de définition et d'administration des processus.

Dans le modèle de référence de la WfMC (voir Figure 2.9), un système de gestion de flux de travail engloberait le moteur de flux et les cinq interfaces avec lesquelles il interagit. Notre évaluation ne se limitera pas aux moteurs de flux de travail. Tout le long du mémoire, on parlera de "moteur de flux de travail" pour indiquer le moteur d'exécution et de "système de gestion de flux de travail" pour indiquer le système composé du moteur et des divers outils de modélisation et d'administration des processus. En effet, la disponibilité et la qualité de ces outils sont déterminants pour faciliter l'adoption du produit dans les organismes de finance sociale et solidaire.

2.2.6 Apports des moteurs de flux de travail

Rationaliser les processus

- La standardisation des méthodes de travail permet une gestion améliorée des processus d'affaires.
- La formalisation des processus assure que toutes les étapes sont correctement suivies grâce aux vérifications de validité et aux audits définis lors de la conception du flux de travail. Ceci permet de réduire les erreurs dues à une mauvaise communication ou à une signification ambiguë des termes.
- L'uniformité des définitions assure une exécution identique pour toutes les instances d'un même processus.
- La constance des processus amène à une plus grande prédictibilité des réponses aux

clients. Il en résulte un meilleur service client.

Augmenter l'efficacité

- L'automatisation de plusieurs processus d'affaires induit l'élimination d'étapes non nécessaires.
- Tous les membres de l'organisme peuvent simultanément exécuter des tâches dans le cadre du flux de travail.
- L'automatisation et la reproductibilité facilitent la mise à l'échelle. L'augmentation du volume des transactions se fait sans augmentation linéaire de l'effectif.
- Les nouveaux employés sont plus facilement formés car les processus les guident à travers les activités correspondant à leurs rôles.
- La manipulation de documents papier et l'acheminement manuel de documents sont réduits.

Disposer d'informations en temps réel

- Les clients et les employés peuvent instantanément connaître l'état d'un item de travail. Les réponses aux questions "où, quand, comment ?" sont fournies par le système de gestion de flux.
- Les gestionnaires peuvent prendre des décisions plus précises étant donné qu'ils savent la situation exacte des processus.
- L'instantanéité du suivi du flux apporte une plus grande agilité dans les interactions.

Mesurer les données sur le flux de travail

- Les audits sur les journaux d'exécution permettent de détecter les goulots d'étranglement dans le flux, le niveau d'efficacité de chaque employé ou encore les données inutiles ou manquantes dans la réalisation des processus.
- L'analyse des processus amène à leur remaniement en concordance avec les besoins d'affaires en perpétuel changement.

2.3 Interactions entre moteurs de règles et moteurs de flux de travail

Les moteurs de règles et les moteurs de flux de travail sont complémentaires. Un langage de définition des processus décrit les processus à un haut niveau d'abstraction. Les règles sont alors utilisées pour implémenter les décisions d'après le contexte du flux de travail. L'automatisation de la prise de décision, qui constitue la fonctionnalité principale d'un système de gestion de règles, ne devrait donc pas faire partie des systèmes de gestion du flux.

2.3.1 Nécessité de séparer les règles des processus d'affaires

La plupart des processus d'affaires, l'accord de prêt étant un bon exemple, contiennent plusieurs points de décision où des règles d'affaires définissent les actions à entreprendre et guident la progression dans le flux de travail. Fréquemment, ces règles sont embarquées dans les processus ou dans du code Java, ce qui peut causer plusieurs problèmes (Bolie et al., 2006) :

- Les règles d'affaires changent plus souvent que les processus. Changer et gérer des règles d'affaires imbriquées est une tâche trop complexe pour la plupart des utilisateurs métier. Ainsi, quand les règles d'affaires changent, les programmeurs doivent s'investir dans la mise à jour du système.
- La plupart des organisations manquent d'un dépôt centralisé de règles. Par conséquent, chaque changement de politique à l'échelle de l'organisation ne peut être appliqué automatiquement pour tous les processus d'affaires.
- Les processus d'affaires ne peuvent pas réutiliser les règles. Ainsi, le personnel informatique finit par concevoir des règles pour chaque processus unique, ce qui amène souvent à des inconsistances et à une redondance.

La meilleure façon d'éviter ces problèmes est d'utiliser un moteur de règles pour séparer les processus des règles, tout en permettant leur interaction. Le fait de déplacer les règles d'affaires en dehors du code applicatif facilite la maintenance. Les gestionnaires métier peuvent modifier les règles lors de l'exécution via une interface graphique. Les règles et

les processus d'affaires peuvent changer indépendamment les uns des autres.

2.3.2 Scénarios d'interaction entre moteurs de règles et moteurs de flux de travail

Ces deux technologies différentes mais connexes peuvent interagir selon divers scénarios [13] :

- Un processus d'affaires atteint un stade où il requiert une décision d'affaires complexe avant de continuer. Il invoque alors un service de règles pour qu'il examine les données et recommande les actions appropriées. La réponse du service de règles est utilisée par le processus d'affaires pour orienter son flux.
- Un système de gestion de règles atteint un point où des données additionnelles sont requises dans le processus de décision. Il initie une instance de processus d'affaires pour alerter les participants concernés et leur faire exécuter les tâches requises. Lorsque l'exécution de l'instance de processus se termine, le système de gestion de règles reprend son exécution là où il s'était arrêté, avec de nouvelles données à sa disposition.
- Si la condition d'une règle est vérifiée, l'action à exécuter consiste à lancer un processus d'affaires complexe. Le processus est instancié et l'exécution des règles se poursuit parallèlement.

Ce second chapitre nous a permis d'expliquer les notions de base des moteurs de règles et de flux de travail. Nous avons également parcouru l'état de l'art ainsi que les technologies et les standards les plus utilisés. Nous avons aussi explicité les bénéfices qu'en tireront les organismes de finance sociale et solidaire. Enfin, les possibilités d'interaction entre moteurs de règles et moteurs de flux de travail ont été évoquées. Maintenant que nous disposons d'une bonne connaissance des moteurs de règles et de flux de travail, il nous est possible de spécifier les exigences auxquelles devra répondre la solution que nous proposerons à la Chaire.

---

13. http ://www.edmblog.com/weblog/2006/02/are_bpms_and_br.html (dernière consultation le 09-11-10)

CHAPITRE III

CRITÈRES D'ÉVALUATION DES MOTEURS DE RÈGLES

Des chercheurs universitaires et des entreprises de consultation spécialisées se sont penchés sur l'évaluation des moteurs de règles et de flux de travail. Dans leurs rapports techniques et leurs articles scientifiques, ils ont conçu des grilles d'évaluation selon plusieurs critères fonctionnels et non fonctionnels. Selon les contextes, certains aspects ont été plus ou moins exhaustivement analysés d'une évaluation à l'autre.

Ce mémoire s'inscrit dans le cadre de la Chaire. Ainsi, concernant l'évaluation des moteurs de règles, une attention particulière sera accordée aux propriétés des langages de définition des règles, aux fonctions facilitant leur expression ainsi qu'à l'accessibilité aux utilisateurs métier. Les fonctionnalités offertes par le moteur sont également exhaustivement évaluées. De même, la qualité des outils de vérification, stockage et versionnage des règles pèsera dans le choix de la solution à recommander.

Au volet des critères non fonctionnels, on évaluera les performances des moteurs, leur courbe d'apprentissage, la documentation offerte ainsi que des critères propres aux logiciels libres.

On décrit chaque critère et on explicite leur pertinence dans le cadre de l'évaluation. On détermine également les moyens de mesurer leur niveau de satisfaction par les produits à tester.

La section suivante envisage quatre scénarios où le produit sélectionné pourrait être utilisé. Les pondérations des critères d'évaluation varient pour chaque scénario, selon leurs impératifs. Par exemple, un organisme de petite taille aura un besoin de performances

d'exécution moins élevé qu'un autre dont le volume de règles d'affaires est important. Des grilles d'évaluation listant les critères et leurs pondérations pour chacun des scénarios dégagés précédemment concluent ce chapitre.
La plupart des critères d'évaluation sont une synthèse de travaux de firmes de consultation tel que (Jenz et Partner, 2000b), (Graham, 2005) et de publications d'experts en moteurs de règles d'affaires. D'autres critères, tel que la prise en charge de l'internationalisation, découlent des besoins de la Chaire.

3.1 Critères fonctionnels

Un critère fonctionnel porte sur une fonctionnalité spécifique définissant ce que le système est supposé réaliser.

3.1.1 Langage de définition des règles

Pour déterminer la qualité des langages de définition de règles offerts par les produits testés, nous évaluons la diversité des moyens d'expression disponibles, l'expressivité du langage, la prise en charge de l'internationalisation et la qualité du lien entre modèle de données et langage de règles.

Prise en charge de plusieurs moyens d'expression des règles

La diversité des contextes fait qu'il est avantageux pour un système de gestion des règles d'affaires de prendre en charge plusieurs moyens d'expression des règles. L'utilisateur, ayant plus de choix, sera alors capable de sélectionner l'approche la plus adéquate à ses préférences.
En effet, il existe différents moyens d'exprimer les règles métier dans un système informatique. On peut utiliser des constructions SI condition ALORS action, des phrases déclaratives ou définitionnelles, poser des contraintes ou encore exprimer les règles en langage naturel.
Le langage d'expression des règles peut soit restreindre la syntaxe soit apprendre à la

machine comment comprendre un large éventail de phrasés. Chaque approche présente des avantages et des inconvénients :

- Une syntaxe formelle bien conçue peut ressembler à du langage naturel et être rapide et facile à manipuler après une étape d'apprentissage. Les vérificateurs syntaxiques peuvent alerter sur des violations de syntaxe ou sur des références à des objets non existants. Cependant, les règles pourraient être incompréhensibles pour une personne non formée et il est impossible de directement prendre des règles exprimées par des experts métier et les recopier telles qu'elles dans le système informatique.
- En revanche, la possibilité d'exprimer les règles en langage naturel permet à un utilisateur métier d'ajouter et de modifier directement les règles. Toutefois, la signification des phrasés en langage naturel doit être explicitée par le créateur de la base de connaissances, ce qui nécessite du temps et de l'effort.

Par ailleurs, les profils des utilisateurs amenés à exprimer les règles d'affaires sont variés. Un utilisateur donné peut être à l'aise avec une approche plus qu'une autre. De plus, chaque approche peut s'avérer plus adéquate pour exprimer un certain type de règles.

Expressivité du langage de définition des règles

Les moteurs de règles d'affaires suivent généralement un modèle déclaratif pour définir les règles. Ce modèle permet de s'affranchir des imbrications et autres structures de contrôle des langages de programmation impératifs tel que Java. Ce style d'écriture est plus adéquat pour exprimer la base de connaissances métier. Toutefois, les phrases déclaratives utilisées dans les moteurs de règles d'affaires sont généralement de la forme SI condition ALORS action. Par exemple :

```
SI le statut du client est important
   et le montant de l'emprunt est inferieur a 2000
   et le client n'a pas un score faible
ALORS approuver l'emprunt
SI le statut du client est important
   et le montant de l'emprunt est inferieur a 2000
   et le client a un score faible
```

```
ALORS ne pas approuver l'emprunt
```

Morgan (Morgan, 2002) recommande un style qui lève l'ambiguïté, explicite les relations, évite une terminologie obscure et supprime la verbosité. Son style se rapproche du langage naturel. Au format précédent, il aurait préféré :

```
Un emprunt est approuvable si
    le statut du client est important
    et le montant de l'emprunt est inferieur a 2000
    a moins que le client ait un score faible.
```

D'autre part, si le produit permet l'expression des règles en langage naturel, il doit refléter la façon dont les utilisateurs métier conçoivent les objets du domaine et la manière avec laquelle ils construisent des phrases pour en parler.
L'expressivité du langage de définition des règles permet d'avoir des règles claires et concises, en conformité avec la réalité du métier et libérées des structures rigides que peuvent imposer certains langages de programmation. Il en résulte une plus grande facilité de conception et de maintenance.

Prise en charge de l'internationalisation

La Chaire ambitionne d'offrir une solution libre pour les organismes financiers de l'économie sociale à travers le monde. Les utilisateurs potentiels du système informatique auront donc des langues, des monnaies et des formatages de date différents.
Un système de gestion de règles qui gère l'internationalisation facilite la réutilisation de la solution dans plusieurs environnements. Ceci donne une dimension internationale au projet et permet d'impliquer une communauté de développeurs et d'utilisateurs à travers le monde. De plus, l'interaction entre des organismes financiers situés dans des environnements différents est facilitée.
Un système de gestion de règles gérant l'internationalisation doit pouvoir paramétrer les valeurs liées à un environnement local et permettre leur transformation sans avoir à altérer d'autres parties des définitions des règles. Par ailleurs, si un langage naturel est

utilisé, il doit être possible de changer facilement la langue avec un besoin minimal de modification des liaisons entre ce langage naturel et le langage d'exécution des règles.

Compréhension du modèle de données

Les règles se réfèrent à un modèle de données, appelé aussi ontologie du domaine. Ce modèle fournit le vocabulaire métier nécessaire à l'expression des règles. Il faut donc pouvoir exprimer les concepts, les relations, le vocabulaire interne et les contraintes métier à partir du langage des règles. L'utilisation d'un langage naturel est particulièrement dépendante de la conception d'un bon lien entre le langage de règles et le modèle de données. À l'instar des outils de mapping objet-relationnel permettant de faire la correspondance entre les tables relationnelles et les objets métier, un système de gestion de règles devrait faciliter la correspondance entre le modèle de données et les règles.
Lors de l'évaluation de ce critère, on vérifiera si le modèle de données est accessible à partir des règles. Des outils de mise en correspondance (mapping) du modèle de données avec le langage de règles constituent un atout dans l'évaluation de ce critère.

3.1.2 Fonctions facilitant l'expression des règles

Cette sous-section évalue la disponibilité d'outils d'aide à l'expression des règles. Ces outils englobent les modèles de règles, l'exécution de procédures et d'algorithmes, les flux de règles et la représentation des connaissances sous forme de tables et d'arbres de décision.

Modèles de règles

Les modèles de règles sont des patrons de conception pour les règles d'affaires. Ils représentent des règles utilisées de manière récurrente avec des variations mineures d'une instance à l'autre. L'utilisateur affecte des valeurs aux variables des modèles en sélectionnant parmi des littéraux, des intervalles, des objets... Un modèle de règles peut être aussi simple qu'une phrase unique avec une seule variable

```
unClient.age > [age minimum].
```

Il peut également être hautement complexe.
Les modèles permettent de réutiliser une même structure en affectant des valeurs uniquement aux sections variables. Cette capacité de réutilisation est particulièrement utile pour des scénarios complexes et récurrents. Les modèles offrent également des structures flexibles permettant un contrôle précis sur ce qui peut et ne peut pas être édité. La facilité de définir des modèles, leur niveau de sophistication et de paramétrisation seront pris en compte dans l'évaluation de ce critère.

Procédures et algorithmes

Certaines connaissances, tel que les formules mathématiques et financières, sont clairement procédurales. Il est par exemple impossible de calculer l'assujettissement à l'impôt sans d'abord connaître les revenus et les dépenses. La représentation de telles connaissances sous forme de règles peut être lourde si le langage de règles n'utilise pas l'approche procédurale.
Un bon moteur de règles devrait permettre l'expression de fonctions sans faire d'appels externes. La cas échéant, il devrait être capable d'invoquer des procédures ou des routines à partir de règles pour qu'elles s'exécutent et lui retournent des résultats.

Flux de règles

Les flux de règles sont un mécanisme natif dans certains moteurs de règles. Ils permettent de spécifier un ordre d'exécution particulier des tâches. Ces dernières peuvent être des ensembles de règles, des fonctions ou encore des modules de flux de règles entiers.
L'orchestration de l'exécution des règles selon un flux peut aussi être implémentée en faisant interagir le moteur de règles avec un moteur de flux : À certaines étapes des processus d'affaires, le moteur de flux appelle le moteur de règles. Ce dernier exécute alors un ensemble de règles pertinentes à cette étape du processus d'affaires et retourne le résultat de son évaluation au moteur de flux. La valeur retournée guide le cheminement

de l'instance du processus d'affaires.
La possibilité d'organiser l'exécution des règles selon des flux est requise dans plusieurs scénarios où un ensemble de règles ne doit être invoqué qu'après une séquence d'activités précise.
Un moteur de règles devrait offrir une gestion native des flux de règles ou, le cas échéant, communiquer avec un moteur de flux de tierce partie afin d'orchestrer le déclenchement des règles.

Tables et arbres de décision

Les connaissances de type "Si/Alors" peuvent être représentées avec des tables ou des arbres de décision. La représentation des règles en arbre peut faciliter le débogage et la communication entre utilisateurs, analystes et développeurs. La définition des règles sous forme de tables décisionnelles est utile principalement lorsque les connaissances existent déjà sous cette forme au sein de l'organisme.
Les systèmes de gestion de règles doivent permettre la représentation des règles en arbre ou en table décisionnelle pour satisfaire ce critère. Ils doivent aussi pouvoir extraire des données à partir de tels formats selon une procédure configurable.

3.1.3 Accessibilité aux experts métier

Cette sous-section évalue les moyens de manipulation des règles fournis aux experts métier.

Accessibilité du langage aux non informaticiens

L'existence d'un langage de règles compréhensible par les utilisateurs métier facilite la définition et la modification des règles. Un langage est considéré accessible aux non informaticiens s'il offre par défaut un format de définition des règles compréhensible par des utilisateurs métier. L'accessibilité du langage est inversement proportionnelle au besoin de formation à sa syntaxe de programmation.

Abstraction des détails d'implémentation

Afin d'être exécutées, les règles doivent être exprimées dans un langage compilable par le moteur de règles. L'abstraction de ce langage technique permet de cacher aux experts métier les détails d'implémentation et de ne leur exposer que la logique métier.
Pour réussir à abstraire les détails d'implémentation, un système de gestion de règles doit automatiquement transformer les règles exprimées en langage d'affaires en code exécutable par le moteur de règles.

Interfaces pour non informaticiens

N'ayant pas de compétences informatiques avancées, les experts métiers interagissent avec le système informatique exclusivement via les interfaces qui leur sont offertes. Un environnement d'édition puissant et ergonomique permet aux non informaticiens d'éditer les règles dans un cadre sécure et productif. Les experts métier doivent être capables de saisir des règles de n'importe quel niveau de complexité à partir d'un environnement, éventuellement web, distinct de l'environnement de développement. L'éditeur doit aussi contraindre les utilisateurs dans leurs choix afin d'éviter de les submerger avec trop d'options et d'assurer une édition sans danger des règles.

3.1.4 Le moteur de règles

Les critères listés dans cette sous-section concernent le moteur d'exécution de règles. Ils évaluent les stratégies implémentées par le moteur d'inférence, la présence de fonctionnalités d'explication du raisonnement et de gestion de l'incertitude, les stratégies de résolution de conflit disponibles ainsi que le niveau d'intégration du moteur à son environnement.

Stratégies du moteur d'inférence

Un moteur d'inférence offre un ou plusieurs moyens d'appliquer les connaissances aux données. Les stratégies les plus communes sont le chaînage avant et le chaînage arrière expliquées au chapitre précédent. Certains moteurs de règles implémentent une approche mixte, prenant en compte les deux stratégies. L'implémentation d'un chaînage avant efficace est difficile car lorsque la base de règles devient volumineuse, les algorithmes naïfs deviennent très lents vu que les faits changent peu dans la mémoire de travail à chaque cycle. Rete est un exemple d'algorithme de chaînage avant efficace.
Les algorithmes d'application des connaissances aux données constituent le cœur des moteurs de règles. De bonnes implémentations amènent donc à de meilleures performances. De plus, certains types de raisonnements ne peuvent être réalisés que si le moteur d'inférence est capable d'appliquer une certaine stratégie. Partir des effets pour deviner les causes n'est par exemple possible qu'en utilisant le chaînage arrière.
Lors de l'évaluation, il sera tenu compte des stratégies prises en charge par chaque moteur de règles ainsi que des éventuelles améliorations propriétaires apportées aux algorithmes de base.

Explication du raisonnement et gestion de l'incertitude

Certains systèmes de gestion de règles sont capables d'expliquer leurs raisonnements, d'incorporer des raisonnements qualitatifs ou encore de gérer l'incertitude. Ils peuvent alors offrir plusieurs conclusions, classées selon une mesure de confiance.
Les systèmes de maintien de la vérité gardent une trace des dépendances entre les assertions et permettent au moteur de règles de rétracter des conclusions de manière conséquente. Ceci permet d'améliorer l'explication de raisonnement et de gérer l'incertitude soulevée lorsque des faits ne sont plus vrais.
L'utilisation de constructions linguistiques signifiant l'incertitude tel que "peut être" ou "il se peut que" pour formuler des conclusions incertaines, l'emploi de facteurs de certitude tel que l'affectation de scores aux différents résultats et l'utilisation d'ensembles

flous (ensembles dont les membres ont une appartenance relative) constituent d'autres techniques de gestion de l'incertitude.
Une implémentation standard de l'explication du raisonnement, fournie avec le système, pourrait s'avérer utile comme aide au débogage.
Le raisonnement sur l'incertitude ajoute de la complexité mais permet de traiter des problèmes plus sophistiqués qui contiennent de la logique floue.

Intégration à l'environnement

Le moteur de règles doit être déployable en tant que composant ou service à l'intérieur d'une application plus large. En effet, il constitue un élément dans la cadre de la solution financière que développe la Chaire. Il est donc nécessaire de pouvoir l'intégrer à une architecture globale et le faire communiquer avec les autres composants du système.
L'intégration se fait à plusieurs niveaux. À la couche présentation, il serait pratique de pouvoir utiliser plusieurs types d'interfaces, selon les environnements des organismes clients. Au niveau de la couche métier, un moteur de règles intégrable aux serveurs d'applications ou pouvant être invoqué comme service web pourra facilement se greffer aux modules existants. Enfin, au niveau couche de données, le moteur devra être capable d'importer et d'exporter des informations de sources externes.
Plusieurs indicateurs permettent de mesurer la capacité d'intégration d'un moteur de règles, notamment :

- Les plateformes et systèmes d'exploitation pris en charge.
- Les étapes nécessaires à l'intégration.
- Le niveau d'automatisation des communications avec les autres composants du système informatique.
- La possibilité d'utiliser différentes implémentations de la couche de présentation.
- L'existence d'intégrations prédéfinies avec des versions récentes de serveurs d'applications.
- La possibilité d'importer et d'exporter des informations selon des formats standards ou aisément convertibles.

Changements dynamiques

Les moteurs de règles permettent de découpler les règles d'affaires du code applicatif. Cette séparation est utile car les règles changent plus rapidement que le reste du système informatique. Pour faire face au changement rapide, il faudrait permettre une modification aisée des règles.
Le système est alors plus réactif et agile. Le moteur de règles est constamment à jour et les coûts de maintenance sont réduits.
Un système satisfait ce critère si :

- La mise à jour des règles nécessite des modifications mineures au code des autres parties du système.
- Il n'est pas nécessaire d'arrêter ou redémarrer l'application de production pour mettre à jour les règles.

3.1.5 Outils de vérification

Vérification de la syntaxe des règles

Avec un langage de définition des règles structuré, il est utile qu'un vérificateur syntaxique surligne les mots clés, les variables et les valeurs avec des couleurs différentes. La vérification de la syntaxe permet de s'assurer en temps réel de la validité du code et de mieux visualiser sa structure globale.

Stratégies de résolution de conflit

Lorsqu'une règle est éligible à être exécutée (sa condition est satisfaite par au moins une combinaison de faits), elle est placée dans l'agenda du moteur de règles. S'il existe dans l'agenda des règles ayant la même priorité, la résolution de conflit est nécessaire pour sélectionner celle qui sera exécutée. Comme l'exécution d'une règle peut modifier la mémoire de travail (l'exécution de la règle A pourrait engendrer la suppression de la règle B de l'agenda), l'ordre d'exécution est parfois déterminant dans les conclusions

produites par le moteur.
Les stratégies de résolution de conflit offertes par le moteur de règle, la possibilité de les modifier et d'en définir de nouvelles sont des facteurs pris en compte dans l'évaluation du critère.

Débogage

Le débogeur est utilisé à l'exécution. Il permet de manuellement contrôler l'exécution des règles grâce à des commandes de débogage comme les points d'arrêts, les points d'inspection et l'exécution pas à pas.
Les points d'arrêt sont utilisés pour suspendre l'exécution du moteur de règles à certains événements ou états d'objets. Le traitement peut être repris à la demande de l'utilisateur.
Les gardes ou *watchpoints* sont utilisés pour suivre de près l'état des objets pendant l'exécution. Le traitement n'est pas suspendu comme c'est le cas pour les points d'arrêt.
L'exécution pas à pas permet de suspendre l'exécution après chaque règle. L'utilisateur peut alors inspecter la mémoire de travail et l'agenda.
L'ensemble de ces outils de débogage facilite la détection et la localisation d'erreurs dans les règles.
Ce critère est satisfait s'il est possible de :
- définir des points d'arrêt et des gardes pour les objets et les événements du système de règles.
- observer des informations en différé, sans arrêter l'exécution du programme.
- effectuer du débogage en pas à pas lors de l'exécution des règles.
- visualiser les étapes d'exécution par lesquelles passe le moteur de règles.

Test des règles

Les tests unitaires et les suites de tests permettent d'exécuter du code et de vérifier que le résultat correspond au comportement requis. Les tests permettent de détecter les erreurs avant que les utilisateurs ne le fassent et de les réparer rapidement et à un faible

coût.

De plus, une fois que les tests sont écrits, ils continuent à vérifier la validité des règles et alertent sur de nouvelles erreurs dès qu'elles surviennent.

Pour satisfaire ce critère, un système de gestion de règles doit disposer d'une prise en charge incorporée des tests et des simulations. Il doit aussi permettre la spécification et l'exécution de suites de tests.

Journalisation et audit

La journalisation permet d'enregistrer les opérations du moteur de règles. Ces données peuvent être utilisées à des fins d'audit. L'audit permet de savoir comment, pourquoi et quand une décision donnée a été prise. Il permet également de dégager des statistiques sur différents aspects de l'édition et de l'exécution des règles.

Une journalisation de qualité devrait permettre :

- La journalisation hiérarchique par granularité des événements.
- Le contrôle du niveau de détails des événements à journaliser (niveau erreur, avertissement, information...).
- La génération de fichiers de journalisation sous formats multiples (texte plat, XML, HTML...).

Pour ce qui est de l'audit, un système de gestion de règles doit présenter les informations journalisées de sorte que l'utilisateur puisse facilement les parcourir et les filtrer afin d'extraire les données dont il a besoin.

3.1.6 Le stockage des règles

Les dépôts de règles stockent les règles d'affaires et permettent de les versionner, de les hiérarchiser et de leur associer des méta-données. À l'instar des systèmes de gestion de bases de données, ils doivent permettre une sélection par critères des règles. Ils doivent également offrir des fonctionnalités d'administration tel que la synchronisation et la réplication.

Versionnage des règles

Il s'agit d'affecter des noms ou des numéros de version uniques à des états uniques d'une même règle.
Le versionnage permet de comparer des définitions de règles. Le versionnage peut s'appuyer sur un système collaboratif de contrôle de version.
Un système implémentant le versionnage des règles est capable de :

- Versionner les fichiers de règles.
- Versionner des projets entiers afin de gérer des versions de publication du système.
- Revenir à une version précédente et exécuter les règles telles qu'elles étaient à un instant passé.

Hiérarchisation des règles

La hiérarchisation permet de répertorier des ensembles de règles dans des paquetages formant une arborescence.
Cette décomposition permet de définir des modules de règles relatifs à des préoccupations distinctes. Ceci facilite l'accès, la réutilisation et la restructuration de blocs de règles cohérents.

Définition de méta-données

Les méta-données sont des "données sur les données". Dans le cadre des règles d'affaires, elles pourraient indiquer l'auteur d'une règle, sa date de création, sa validité...
Les méta-données apportent un contexte plus exhaustif aux règles en fournissant toute sorte d'informations pertinentes les concernant. Ces informations pourraient être utilisées, par exemple, pour effectuer des recherches avancées de règles.
Un dépôt de règles de qualité permet de stocker des méta-données de n'importe quel type sur les règles.

Recherche par critères

Il s'agit de trouver les règles qui contiennent certains mots clés, qui ont été écrites par un certain utilisateur ou définies à une certaine date, etc.
La recherche de règles par critères est une fonctionnalité importante, au même titre que la sélection d'enregistrements dans une base de données. Elle permet de rapidement localiser l'information en affectant des filtres de sélection.
Le critère est satisfait si on peut :
- effectuer des recherches sur les règles selon un terme ou une phrase spécifique.
- effectuer des recherches selon la date d'édition ou la source d'édition ou une autre méta-donnée des règles.
- sauvegarder des requêtes pour une utilisation future.

Administration des dépôts

L'administration des dépôts contribue à la maintenance des règles. Elle facilite également leur partage et leur réutilisation.
Les fonctionnalités d'administration dont on évaluera l'existence sont :
- La synchronisation des dépôts.
- L'importation / exportation de dépôts.
- La réplication, sauvegarde et restauration des dépôts.
- La possibilité de stocker les règles selon plusieurs mécanismes (bases de données, fichiers plats, répertoires LDAP (Lightweight Directory Access Protocol)...).

3.2 Critères non fonctionnels

Les critères non fonctionnels évaluent des aspects globaux relatifs à l'exploitation du produit, plutôt que l'existence de fonctionnalités et de comportements spécifiques.

3.2.1 Performance et coûts en ressources

Performance des moteurs de règles d'affaires

Les moteurs de règles étant utilisés pour résoudre des problèmes complexes, leurs performances sont importantes. Une bonne conception des règles constitue le premier critère pour avoir un système performant. Cependant, les algorithmes et stratégies utilisés par le moteur affectent les temps de traitement. Les bancs d'essai (benchmarks) comparent les performances de quelques uns des moteurs de règles les plus communs.

Banc d'essai Miss Manners : Manners réfère au problème de trouver un arrangement acceptable de placement d'invités autour d'une table. La solution doit s'assurer que chaque invité est assis à côté d'une personne du sexe opposé qui partage avec elle au moins un centre d'intérêt. Pour comparer les performances des moteurs de règles, l'ensemble des règles de Manners sont traduites dans le langage de règles du moteur à tester.

Le banc d'essai va tester l'agenda et les noeuds beta du réseau Rete. Tous les faits qui coïncident (deux personnes de sexe opposé avec un centre d'intérêt en commun) seront propagés dans le réseau. De nombreuses correspondances partielles ne vont pas aboutir à des correspondances complètes.

À titre d'exemple, la société Illation a réalisé des tests avec Miss Manners [1] en prenant quatre ensembles de données de test équitablement espacés (16, 32, 64 et 128 invités à placer). Chaque ensemble distribue uniformément les centres d'intérêt sur les invités. Il est à noter que plusieurs moteurs de règles d'affaires ont été optimisés pour ce banc d'essai, ce qui fait qu'il n'est plus très utile pour évaluer le niveau de performance réel des moteurs.

Banc d'essai Waltz : Waltz est un autre banc d'essai populaire. Il réfère à l'interprétation en trois dimensions du dessin d'une ligne en deux dimensions en assignant des labels aux lignes à deux dimensions (abscisse et ordonnée). Comme le test Manners, Waltz teste l'agenda et les noeuds beta du réseau Rete.

1. http ://illation.com.au/benchmarks/manners.html (dernière consultation le 09-11-10)

Banc d'essai WaltzDB : WaltzDB est une version plus générale de Waltz conçue pour gérer des jonctions de quatre à six lignes alors que Waltz ne gère que des jonctions de deux à trois lignes. WaltzDB est considéré comme un des meilleurs outils actuellement disponibles pour comparer les moteurs de règles d'affaires.
L'université du Texas à Austin, entre autres, héberge le code des suites de test Waltz et WaltzDB [2].

Limitations des bancs d'essai

Les bancs d'essai cités ci-haut ont de nombreuses limitations :

- Ils traitent un ensemble restreint de caractéristiques et testent principalement une fonctionnalité spécifique prise en charge par la plupart, mais pas par tous les moteurs.
- Manners n'est pas assez explicite sur l'approche à suivre pour résoudre le problème. Les résultats des bancs d'essai dépendent significativement de paramètres non documentés. Il est facile de "tricher" dans les tests de Manners en insérant une condition additionnelle dans une des règles au bon endroit et permettre ainsi au moteur de trouver la conclusion correcte en beaucoup moins de temps que prévu.
- L'exactitude des solutions produites par les moteurs n'est pas vérifiée, ce qui signifie qu'elle pourrait être sacrifiée pour avoir plus de rapidité. Ceci est particulièrement vrai pour Waltz et WaltzDB qui, en général, ne vérifient pas les solutions produites.
- Les bancs d'essai testent les pires cas d'inférence et d'utilisation d'agenda de Rete, ce qui représente rarement le cas réel dans des applications bien conçues.

Pour toutes ces raisons, on ajoutera aux résultats des bancs d'essais une évaluation plus globale des performances. Les retours d'informations des utilisateurs, publiés sur les forums et les listes de courriel des produits constituent un bon indicateur des performances des moteurs dans des environnements de production.

---

2. http ://www.cs.utexas.edu/ftp/pub/ops5-benchmark-suite/ (dernière consultation le 09-11-10)

Coûts en ressources matérielles

La solution informatique de la Chaire, dont fera partie le moteur de règles à recommander, est destinée à des organismes de l'économie sociale à travers le monde. Ces organismes disposent généralement de moyens logistiques plus ou moins limités. Il est donc important que le système qui leur soit proposé n'ait pas d'exigences techniques rédhibitoires.
Lors de l'évaluation des moteurs de règles, la consommation en mémoire vive, en espace disque et en temps de calcul sera prise en compte.

3.2.2 Prise en main

Documentation

La facilité de prise en main dépend de la qualité de la documentation fournie. Elle contribue à une utilisation optimale du produit et à une bonne compréhension de son fonctionnement. Ceci est d'autant plus vrai que les systèmes de gestion de règles constituent une notion relativement récente et pas encore largement comprise.
Ce critère évalue la présence et la qualité des exemples, tutoriels et fichiers d'aides couvrant les divers fonctionnalités des produits. Il est également important que l'aide soit disponible aussi bien pour l'utilisateur novice qu'expérimenté. Enfin, il est souhaité que la documentation inclut des informations sur les meilleures pratiques.

Convivialité

Ce critère englobe le temps requis pour l'installation des principales fonctions du produit, la facilité de l'installation ainsi que la convivialité des interfaces utilisateur.
Un produit qui s'installe rapidement et facilement et qui présente des interfaces utilisateur ergonomiques a de meilleures chances d'être adopté par les utilisateurs métier.

Courbe d'apprentissage

La courbe d'apprentissage se rapporte au temps nécessaire pour se familiariser avec le système de gestion de règles et être capable d'en faire une utilisation optimale. Une courbe d'apprentissage faible indique que le système est rapidement pris en main. Ceci permet une adoption rapide et facilite l'intégration du système de gestion de règles aux systèmes informatiques existants.
La facilité d'utilisation du produit pour les utilisateurs non formés, ainsi que les compétences pré-requises pour une utilisation efficace de l'outil permettent d'évaluer la courbe d'apprentissage.

3.2.3 Évaluation sur l'aspect logiciel libre

Les moteurs de règles autant que les moteurs de flux de travail qui seront évalués appartiennent exclusivement au monde du logiciel libre. Ils obéissent à des codes caractéristiques de cet univers et sont jugés selon des facteurs qui leur sont propres.

Communauté

La communauté a une importance considérable dans les projets libres car elle effectue la majorité des tests du produit et fournit des retours d'information appréciables. Au lieu d'utiliser des ressources financières pour effectuer de vastes opérations d'assurance qualité, comme le font les produits propriétaires, les projets libres possèdent leurs communautés comme ressource.
Plus le nombre de personnes intéressées par un projet est grand, plus ses chances d'être actif et de continuer sont grandes. Une communauté importante et active est aussi un bon indicateur de l'acceptation du logiciel : s'il n'était pas de bonne qualité, il n'y aurait pas autant de monde qui serait concerné par son développement (Duijnhouwer et Widdows, 2003).

Fréquence de publication

Bien qu'un projet puisse se stabiliser avec le temps, il ne devrait jamais devenir complètement statique. La fréquence des publications et l'importance des changements qu'elles apportent illustrent le progrès réalisé par les développeurs. Ceci constitue un bon indicateur de leur niveau d'implication dans le projet.

Longévité

La longévité d'un produit est une mesure de sa période d'existence. Elle constitue un indicateur de sa stabilité et de ses chances de continuation. Un nouveau produit contient généralement beaucoup de bogues (Golden, 2004). Ceci dit, un produit très ancien pourrait être obsolète. Il est nécessaire donc de vérifier, pour ce type de produits, qu'il existe une communauté actuellement active. Ceci indique que le produit a atteint un bon niveau de maturité.

Licence

La licence d'un logiciel peut avoir des conséquences non souhaitables selon l'utilisation que l'on veut en faire. Si l'on planifie de modifier et de redistribuer le logiciel sous une forme mais qu'on ne désire pas distribuer le code source, une licence de type "copyleft" tel que GPL n'est pas recommandée. Dans le cadre de la Chaire, les moteurs de règles et de flux de travail ne seront pas revendus aux organismes de finance sociale. Leurs licences devraient préférablement être parmi celles reconnues dans le monde du logiciel libre et être cohérentes avec l'utilisation voulue du produit.

Assistance communautaire

Un produit logiciel peut avoir :

- une assistance usager, où des questions sur l'utilisation du produit sont traitées.
- une assistance traitant les défaillances, où les problèmes du logiciel sont résolus.

Une assistance de qualité est un indicateur de l'implication des développeurs. L'existence d'un logiciel de suivi des bogues (bug tracker) dédié et la fréquence de son utilisation par les utilisateurs et les développeurs du logiciel permettent d'évaluer la qualité de l'assistance disponible (van den Berg, 2005).

Assistance professionnelle

À côté de l'assistance communautaire, un produit libre peut offrir un service d'assistance professionnel payant. Par ailleurs, des développeurs professionnels rémunérés sont assignés au développement et à la maintenance du projet à temps plein, par l'entreprise partenaire.
L'assistance professionnelle est un signe que le produit est mature et qu'il jouit d'une utilisation conséquente dans le cadre professionnel.
Il s'agit d'un gage de fiabilité qui devrait encourager les organismes frileux à adopter le produit pour des tâches critiques.

3.3 Pondération des critères d'évaluation selon l'environnement

L'importance des différents critères d'évaluation varie d'un organisme à l'autre en fonction de sa taille, de la composition de son personnel, de la nature des services qu'il offre et de ses impératifs et préoccupations.
Afin de mettre en contexte les critères d'évaluation, on définit quatre scénarios d'utilisation, illustrant des prototypes d'environnement d'entreprises. Les notes par critère sont les mêmes dans les quatre scénarios, mais les poids donnés à chaque critère varient d'un scénario à l'autre selon des impératifs culturels et techniques dans chaque situation.
Il est à noter que les scénarios illustrent des orientations globales, chaque organisme peut donc personnaliser les poids des critères comme il l'entend, afin de représenter au mieux ses besoins spécifiques.
Enfin, il est possible de combiner plusieurs scénarios. Par exemple, un organisme pourrait avoir des règles volumineuses et sophistiquées (scénario 1), impliquer plusieurs départements et type d'utilisateurs dans le système de gestion de règles (scénario 2)

et posséder un département informatique interne important (scénario 3). En cas de combinaison de scénarios, on suggère de prendre le poids le plus élevé attribué à un critère dans les différents scénarios applicables à l'organisme : si un scénario est applicable alors les critères qui lui sont essentiels doivent l'être dans l'évaluation.

3.3.1 Critères invariablement importants

Certains critères sont importants quelque soit l'environnement de l'organisme client. En effet :

- Le langage de définition des règles doit être expressif.
- Le langage de règles doit comprendre le modèle de données.
- Étant donné qu'il s'agit d'organismes financiers, il est nécessaire de pouvoir appeler des procédures et des algorithmes (qui implémentent des fonctions du domaine financier) à partir des règles.
- Les aspects liés à la nature libre de la solution tel que la qualité de la communauté, la fréquence de publications, la longévité et la licence du produit sont toujours pertinents.
- Les fonctions de journalisation et d'audit sont nécessaires pour garder une trace des exécutions.

3.3.2 Scénario 1 : Règles volumineuses et sophistiquées

Même si les effectifs et l'infrastructure dont disposent les corporations financières capitalistes sont généralement plus importants que ce qui existe dans les entreprises de finance sociale, le volume de règles n'y est pas forcément plus important. En effet, le portefeuille de produits offert en finance sociale est large et varié, allant de l'assurance automobile et habitation, aux livrets d'épargne, en passant par les chèques déjeuner, les cartes de retrait ou encore les prêts et les placements. De plus, certains organismes de finance sociale disposent d'une infrastructure importante. Si les règles d'affaires liées à la satisfaction des actionnaires sont absentes de ce genre d'entreprises, d'autres règles visant à assurer la nature solidaire et équitable des opérations viennent se rajouter.

Ce scénario envisage un organisme de l'économie sociale disposant d'un volume important de règles d'affaires sophistiquées.

Dans ce cadre,

- Il est important de pouvoir factoriser les éléments communs aux différentes règles d'affaires et de réutiliser des modèles récurrents via des modèles de règles.
- La performance du moteur de règles ainsi que les types de stratégies disponibles et leur qualité d'implémentation sont essentiels.
- L'explication du raisonnement et la gestion de l'incertitude permettent de représenter des règles sophistiquées.
- Avec un grand nombre de règles, la gestion des stratégies de résolution de conflits est importante.
- Le stockage dans des dépôts de règles et les fonctions de versionnage et de recherche par critères sont importants.
- Enfin, un tel environnement suppose un changement fréquent des règles et une utilisation continue du système. Ceci amène à mettre l'emphase sur une modification aisée de la mémoire de production.

3.3.3 Scénario 2 : Variété des utilisateurs

Un organisme financier peut disposer de plusieurs départements responsables des différents services offerts aux clients. Le système de gestion des règles peut impliquer plusieurs types d'utilisateurs ayant des formations diverses. L'organisme peut également avoir des succursales localisées dans des environnements différents (parties anglophone et francophone du Canada) ou encore interagir avec des partenaires internationaux.

Ce scénario met l'accent sur la variété des utilisateurs du système et de leurs environnements.

- La prise en charge de plusieurs moyens d'expression des règles permet de couvrir plusieurs approches de conception.
- La prise en charge de l'internationalisation facilite l'adaptation du système à plusieurs environnements.

- La possibilité de définir des flux de règles est utile pour guider le flux de travail et diriger les activités assignées aux différents acteurs.
- La gestion des stratégies de résolution de conflits permet de s'assurer que les entrées fournies par différents services de l'organisme ne sont pas en conflit.
- La hiérarchisation des règles dans les dépôts de stockage permet de les structurer dans des paquetages cohérents.
- La possibilité d'utiliser les tables et les arbres décisionnels offre un support d'échange d'informations compréhensible par des utilisateurs de formations diverses.
- Les modèles de règles facilitent la normalisation des règles pour tous les utilisateurs.
- L'accessibilité aux utilisateurs métier permet de les impliquer dans la définition des processus.

3.3.4 Scénario 3 : Existence de compétences informatiques internes importantes

Dans ce troisième scénario, l'organisme dispose d'un département informatique important. Des informaticiens sont employés à temps plein et sont disposés à prendre en charge le système de gestion des règles. Il existe un engagement fort envers une architecture technique. Le système de gestion de règles doit être intégré à plusieurs bases de données existantes.

Les utilisateurs métier sont disponibles pour l'explication des connaissances mais n'ont pas le temps ou ne sont pas disposés à créer eux mêmes les règles. Ils les maintiennent via des applications personnalisées, développées par le département informatique, lorsque nécessaire.

Dans ce genre d'environnement,

- L'équipe d'informaticiens doit disposer d'outils de test et de débogage des règles.
- La présence d'un système informatique important déjà déployé et contenant les données métier fait que le système de gestion de règles doit s'intégrer facilement à l'environnement.
- La gestion des tables décisionnelles accélère l'import de données existantes sous ce format dans l'organisme.

- Les arbres de décision et l'explication du raisonnement facilitent le débogage des règles par les développeurs.

3.3.5 Scénario 4 : Implication des utilisateurs métier

Ce scénario illustre un environnement où le personnel non technique est enthousiaste pour s'investir dans la création des règles, avec l'aide des collègues informaticiens. Les analystes métier ne sont pas forcément compétents en programmation mais comprennent bien les règles d'affaires. Le département informatique interne à l'organisme est relativement petit.

Dans ce cadre :

- L'accessibilité aux experts métier et la facilité de prise en main sont capitales.
- Les modèles de règles assistent dans la définition des règles.
- La possibilité de concevoir les règles dans des tables et des arbres décisionnels est judicieuse si ce format est utilisé par les experts métier.
- La vérification de la syntaxe est requise si le langage de définition des règles est formalisé (si ce n'est pas un langage naturel).
- La présence d'une assistance communautaire et professionnelle permettent d'aider les experts métier.
- Les ressources logistiques étant limitées, les coûts en ressources matérielles de la solution à implémenter sont pris en considération.

Maintenant que les critères et les principaux environnements d'utilisation des systèmes de gestion de règles sont définis, on établit les grilles d'évaluation.

3.4 Grilles d'évaluation des moteurs de règles

La grille ci-dessous récapitule l'ensemble des critères d'évaluation définis dans le cadre de notre évaluation des moteurs de règles. Chaque critère se voit attribuer un poids en fonction du scénario d'utilisation.

Les valeurs des poids sont données sur une échelle de 0 à 5. 0 indique que le critère est sans importance dans cet environnement, 1 signifie qu'il est faiblement important,

2 qu'il est moyennement important, 3 qu'il est important, 4 qu'il est très important et 5 qu'il est essentiel.

L'ordre des colonnes des scénarios S1, S2, S3 et S4 correspond à leur ordre de définition à la section précédente :

- S1 : Règles volumineuses et sophistiquées
- S2 : Variété des utilisateurs
- S3 : Existence de compétences informatiques internes importantes
- S4 : Implication des utilisateurs métier

Attribut	S1	S2	S3	S4
A. Critères fonctionnels				
Langage de définition des règles				
1. Prise en charge de plusieurs moyens d'expression des règles	2	4	2	3
2. Expressivité du langage de définition des règles	5	5	5	5
3. Prise en charge de l'internationalisation	4	5	3	3
4. Compréhension du modèle de données	5	5	5	5
Fonctions facilitant l'expression des règles				
5. Modèles de règles	4	4	3	4
6. Procédures et algorithmes	4	4	4	4
7. Flux de règles	3	4	3	3
8. Tables et arbres de décision	3	4	4	4
Accessibilité aux experts métier				
9. Accessibilité du langage aux non informaticiens	3	4	2	5
10. Abstraction des détails d'implémentation	3	3	2	5
11. Interfaces pour non informaticiens	3	4	2	5
Le moteur de règles				
12. Stratégies du moteur d'inférence	5	4	3	3
13. Explication du raisonnement et gestion de l'incertitude	4	2	4	2
14. Intégration à l'environnement	3	3	5	4

15. Changements dynamiques	4	2	3	3
Outils de vérification des règles				
16. Vérification de la syntaxe des règles	4	4	4	5
17. Stratégies de résolution de conflit	5	5	3	5
18. Débogage	4	3	5	3
19. Test des règles	4	3	5	3
20. Journalisation et audit	4	4	4	4
Le stockage des règles				
21. Versionnage des règles	5	4	3	4
22. Hiérarchisation des règles	4	5	3	3
23. Définition de méta-données	3	3	3	3
24. Recherche par critères	5	4	3	4
25. Administration des dépôts	3	3	3	3
B. Critères non fonctionnels				
Performance et coût				
26. Performance des moteurs de règles	5	4	4	3
27. Coûts en ressources matérielles	3	3	2	5
Prise en main				
28. Documentation	4	4	4	5
29. Convivialité	3	3	2	5
30. Courbe d'apprentissage	4	4	4	5
Évaluation sur l'aspect logiciel libre				
31. Communauté	4	4	4	5
32. Publications	4	4	4	4
33. Longévité	4	4	4	4
34. Licence	5	5	5	5
35. Assistance communautaire	4	4	3	5
36. Assistance professionnelle	3	3	2	5

Tableau 3.1: Grille d'évaluation des moteurs de règles

Ce chapitre a défini l'ensemble des critères fonctionnels et non fonctionnels qui seront utilisés dans l'évaluation des moteurs de règles. Afin de prendre en compte la variété des contextes de déploiement, on a décrit quatre environnements typiques d'organismes clients. Chaque environnement a mis l'emphase sur certains critères de l'évaluation. Enfin, la grille d'évaluation a récapitulé l'ensemble des critères d'évaluation. Chaque critère s'est vu attribuer un poids pour chacun des quatre scénarios d'utilisation.
L'évaluation des moteurs de flux de travail suivra globalement la même structure. Certains critères relatifs à la facilité d'accès, aux performances et coûts ou encore à la nature libre des moteurs de règles seront tout aussi pertinents pour évaluer les moteurs de flux de travail.

CHAPITRE IV

CRITÈRES D'ÉVALUATION POUR LES MOTEURS DE FLUX DE TRAVAIL

Les critères pour l'évaluation des moteurs de flux de travail sont une synthèse des travaux exhaustifs de l'initiative des patrons de flux de travail (Russell et al., 2006), (Russell et al., 2004a), (Russell et al., 2004b) et (Russell, van der Aalst et ter Hofstede, 2006) ; d'autres publications sur le sujet tel que (Ader, 2004), (Jenz et Partner, 2000a) et de critères propres au contexte de la chaire (logiciel libre, environnement international, organismes d'économie sociale et solidaire).

On entame ce chapitre par une présentation des patrons de flux de travail. Viennent ensuite les critères d'évaluation qui sont regroupés en critères fonctionnels et critères non fonctionnels, comme pour les moteurs de règles. Certains critères utilisés pour les moteurs de règles sont repris s'ils sont applicables et pertinents pour l'évaluation des moteurs de flux. La troisième section du chapitre dresse des prototypes d'environnements typiques où le moteur de flux de travail est destiné à être utilisé. Enfin, une grille d'évaluation récapitule l'ensemble des critères en affectant des pondérations différentes à chacun selon le scénario.

4.1 Les patrons de flux de travail

Un des efforts les plus remarquables pour poser des critères d'évaluation des moteurs de flux de travail est celui entrepris par les professeurs Wil Van der Aalst et Arthur ter Hofstede, respectivement de l'Université de Technologie d'Eindhoven aux Pays-Bas et

de l'Université de Technologie du Queensland en Australie. Les chercheurs ont mis en place des *workflow patterns* dont ils mesurent le degré d'application pour les différents moteurs de flux de travail sur le marché. Le travail a notamment été motivé par l'observation que, bien que plusieurs systèmes de flux de travail soient basés sur des concepts fondamentaux similaires, ils diffèrent nettement dans leur pouvoir expressif et l'étendue des concepts qu'ils sont capables de saisir [1].

Les patrons listent les différentes opérations que peut modéliser un langage de description de flux de travail. Ils sont nombreux et regroupés par perspectives : perspective des contrôles de flux, perspective des données, perspective des ressources et perspective de la gestion des exceptions.

Les patrons permettent de jauger la pertinence d'un système de flux de travail en dégageant ses zones de force et de faiblesse dans l'expression des différents aspects du flux de travail. Signe de leur justesse, ces patrons ont inspiré et influencé bon nombre de systèmes tel que jBPM, OpenWFE et bizAgi [2]. Ils ont également été utilisés comme critères de sélection lors de choix critiques tel que la sélection d'un système de flux de travail pour le département de justice néerlandais [3]. Par ailleurs, ils ont été souvent cités dans des articles scientifiques [4].

La *Workflow Patterns initiative* a publié des évaluations de la conformité des standards de flux de travail (BPMN, XPDL, BPEL...) à ces patrons. Elle a aussi évalué leur implémentation pour les moteurs de flux de travail commerciaux. Dernièrement, trois moteurs de flux de travail libres (jBPM, OpenWFE et Enhydra Shark) ont fait l'objet d'une évaluation par patrons de flux de travail(Wohed et al., 2009a).

Un système de notation simple et précis a été adopté pour les évaluations :

- + signifie que le patron est directement pris en compte par le langage de définition

1. (Russell et al., 2006)

2. http ://www.workflowpatterns.com/impact/system.php (dernière consultation le 09-11-10)

3. http ://www.workflowpatterns.com/impact/selection.php (dernière consultation le 09-11-10)

4. http ://www.workflowpatterns.com/impact/citations.php (dernière consultation le 09-11-10)

de processus.

- +/- signifie qu'il n'est pas directement pris en compte, mais qu'il peut être réalisé en invoquant du code Java par exemple.
- - signifie que l'expression du patron résulte en "du code ou des diagrammes spaghetti"

Le tableau 4.1 illustre une portion des résultats des tests établis dans les évaluations (Wohed et al., 2009b)

Patron	Produit		
	jBPM	OpenWFE	Enhydra Shark
Choix exclusif	+	+	+
Convergence simple	+	+	+
Choix multiples	-	+/-	+
Convergence structurée synchronisée	-	-	-
Discriminateur structuré	-	+	-

Tableau 4.1 Extrait des évaluations (Wohed et al., 2009b)

Dans l'évaluation à venir, seulement un sous-ensemble des patrons fera partie des critères. En effet, les patrons visent à répertorier l'ensemble des fonctionnalités et scénarios d'utilisation relatifs aux flux de travail. Il est donc nécessaire d'écarter l'évaluation de certains patrons non pertinents dans le cadre de l'utilisation visée par la Chaire.
Par ailleurs, les patrons pris individuellement vérifient l'existence d'un aspect très précis et techniquement cadré ; de telle sorte qu'une fonctionnalité se trouve décomposée en plusieurs variantes avec des changements mineurs d'un patron à l'autre. Cette décomposition n'est pas pertinente dans le contexte de notre travail où on synthétisera les patrons selon des fonctionnalités générales et non un comportement très ciblé.
Trois groupes de critères majeurs de notre grille personnalisée découlent des patrons de flux : les critères relatifs au contrôle du flux, à la distribution des items de travail et aux données.

Concernant l'attribution de notes, il sera tenu compte des résultats publiés par l'initiative des patrons de flux de travail. Par exemple, pour le système jBPM, son évaluation ainsi que celle du standard XPDL qu'il utilise seront toutes deux considérées. Au cas où l'évaluation publiée est obsolète, seule notre évaluation personnelle est prise en compte.

4.2 Critères fonctionnels

Les critères fonctionnels sont regroupés en critères relatifs au langage de définition des processus, au contrôle du flux de travail, à la distribution des items de travail, à la représentation et à l'utilisation des données, à l'accessibilité aux utilisateurs métier, au moteur d'exécution des flux, au stockage des processus et aux outils de vérification.

4.2.1 Langage de définition des processus

Cette section évalue la présence de plusieurs moyens d'expression des processus, la prise en charge de l'internationalisation et la compréhension du modèle de données.

Prise en charge de plusieurs moyens d'expression

Les différentes activités peuvent être exprimées avec un langage mieux qu'un autre. Les flux de travail faisant intervenir des acteurs de plusieurs spécialités, il est judicieux d'offrir une palette de langages afin de pouvoir sélectionner le plus adéquat selon la nature de l'opération à réaliser.

Prise en charge de l'internationalisation

Un système de gestion du flux de travail qui gère des entités provenant de plusieurs environnements doit prendre en charge l'internationalisation. À titre d'exemple, un organisme financier canadien francophone pourrait avoir des échanges avec un organisme américain ou canadien anglophone. Le formatage de dates, la langue de communication avec les acteurs humains et la monnaie utilisée pour les opérations financières peuvent alors varier. Les variables propres à l'environnement doivent être paramétrables. La mo-

dification de l'environnement doit se faire indépendamment de la logique de flux et avec une intervention minimale sur le code applicatif.

Compréhension du modèle de données

Ce critère évalue la facilité de manipulation d'entités du domaine de données à partir de l'environnement de définition des flux de travail. Les changements d'entités du domaine doivent être détectés dans l'environnement de définition de flux. Si les données sont stockées dans une base de données, la création de la correspondance (*mapping*) avec ou sans un schéma de base de données par le système de gestion de flux est un atout.

4.2.2 Contrôle du flux de travail

Les patrons de contrôle de flux (Russell et al., 2006) regroupent les moyens de contrôle du flux d'exécution des processus. Ils sont subdivisés en contrôles élémentaires et avancés, en contrôles basés sur l'état du processus, en structures d'itération et en contrôles d'annulation. La gestion d'instances multiples et le démarrage déclenché par des signaux externes font également partie des contrôles de flux.

Contrôle élémentaire du flux

Il s'agit des blocs de construction fondamentaux du contrôle de la séquence d'exécution d'un flux de travail.

- Séquence de tâches : Une tâche est activée après complétion de la tâche précédente dans la même instance de processus.
- Divergence parallèle : Divergence d'une branche en au moins deux branches parallèles. Les branches s'exécutent de manière concurrente.
- Synchronisation : Convergence d'au moins deux branches parallèles en une seule branche qui n'est activée que lorsque toutes les branches parallèles ont atteint le point de convergence.
- Choix exclusif : Divergence d'une branche en au moins deux branches. Au point de

divergence, un mécanisme de sélection active une seule des branches subséquentes.
- Convergence simple : Convergence d'au moins deux branches parallèles en une seule branche. L'exécution se poursuit pour chaque branche après le point de convergence sans attendre que les autres branches ne l'atteignent.

Contrôle avancé du flux

Ces contrôles ne sont pas toujours exprimables par les langages de définition de flux, bien qu'ils soient communément utilisés dans des scénarios métier.
- Choix multiples : Divergence d'une branche en au moins deux branches. Au point de divergence, un mécanisme de sélection active un sous-ensemble de branches subséquentes.
- Convergence synchronisante : Convergence d'au moins deux branches parallèles, provenant d'une divergence en choix multiples, en une seule branche. L'exécution se poursuit soit lorsque toutes les branches parallèles ont atteint le point de convergence soit lorsqu'il est impossible qu'une branche qui n'ait pas encore atteint le point de convergence le fasse.
- Convergence multiple : Similaire à la convergence simple à la différence que chaque branche qui atteint le point de convergence multiple poursuit son exécution indépendamment des autres branches activées lors du choix multiple.
- Discriminateur : Convergence d'au moins deux branches parallèles en une seule branche. La branche qui atteint en premier le point de convergence active l'exécution de la suite du flux. L'arrivée de toutes les autres branches au point de convergence n'a aucune incidence sur les tâches subséquentes.

Contrôle basé sur l'état du processus

Ces contrôles reflètent les cas où l'état actuel d'un processus (données associées à l'exécution en cours) est déterminant dans la séquence de tâches qu'il exécute.
- Choix différés : Point de divergence où une des branches est choisie lors de l'exécution selon des données de l'environnement. À la conception, toutes les branches sont activables.

- Routage parallèle interfolié : Capacité de définir un ordre partiel pour un ensemble de tâches. Chaque tâche de l'ensemble ne s'exécute qu'une fois dans n'importe quel ordre conforme à l'ordre partiel. Par exemple, dans le processus de traitement d'une demande d'emprunt, la tâche "vérifier la validité de la demande" doit s'exécuter avant la tâche "évaluer la demande". La tâche "imprimer un accusé de réception de la demande" peut être exécutée à n'importe quel moment.
- Arrivée à un stade (Milestone) : Une tâche ne peut être activée que lorsque l'instance du processus auquel elle appartient atteint un certain point qui se trouve généralement dans une branche parallèle. Si l'instance du processus a dépassé ce point, la tâche ne peut plus être activée. Ainsi, une branche d'un processus ne peut avancer que lorsque l'autre branche a atteint un stade spécifique. Cette fonctionnalité permet de définir une exécution conditionnelle d'une tâche ou d'un ensemble de tâches.

Itération dans le flux

Ces contrôles saisissent des comportements répétitifs dans un flux de travail.

- Boucles : Capacité de répéter l'exécution d'une tâche ou d'un ensemble de tâches via un opérateur de bouclage qui possède un point d'entrée et un point de sortie uniques. L'opérateur peut décider de répéter la/les tâche(s) au début ou à la fin de chaque boucle (similaire à *while...do* et *repeat...until*).
- Cycles arbitraires : Capacité de représenter des cycles dans la définition des processus. Une des branches émanant d'une même tâche permettrait de répéter le flux à partir d'une certaine tâche et une autre permettrait de poursuivre l'exécution sans répétition. Les cycles arbitraires ne nécessitent pas de définir des opérateurs de bouclage.
- Récursivité : Capacité pour une tâche d'invoquer une nouvelle instance du processus dont elle fait partie. Certains flux de travail peuvent être réalisés de façon plus simple avec des appels récursifs qu'avec des boucles.

Annulation d'éléments dans le flux

Ces contrôles permettent d'annuler l'exécution d'éléments dans des instances de processus. Ceci est utile pour gérer les exceptions et éviter d'exécuter des opérations invalidées.

- Annuler une tâche : Une tâche activée est retirée avant le début de son exécution. Si elle a déjà démarré, elle est désactivée et l'instance de processus à laquelle elle appartient est suspendue ou retirée dès que possible.
- Annuler une région : Annulation d'un ensemble de tâches. Les tâches ne se suivent pas forcément et peuvent se trouver à des endroits différents du processus. Cette fonctionnalité est utile si un ensemble limité de tâches non reliées doivent être retirées d'un processus.
- Annuler une instance de processus : Annulation de l'instance de processus au complet et de ses sous-instances. Cette fonctionnalité est utile si une condition nécessaire à la poursuite de l'instance n'est plus vraie. Par exemple, une instance d'accord de prêt est annulée si le client ne désire plus obtenir de prêt.

Gestion d'instances multiples

Ces contrôles permettent de créer plusieurs instances d'une tâche à l'intérieur d'une même instance de processus. Ces instances de tâches sont indépendantes les unes des autres et s'exécutent toutes dans le cadre de la même instance de processus. Par exemple, un employé reçoit un ensemble de demandes d'emprunt à traiter. Pour chaque demande, il doit produire un acquittement. La tâche "produire un acquittement" est instanciée plusieurs fois à l'intérieur d'une unique instance du processus "traiter liste des demandes d'emprunt".

- Instances multiples connues dès la conception : Le nombre d'instances multiples à créer est fixé avant l'instanciation de processus.
- Instances multiples connues à l'exécution : Le nombre d'instances de la tâche dépend de facteurs connus à l'exécution tel que la disponibilité de ressources, l'état de l'instance de processus... Par exemple, le processus "auditer client" implique l'exécution

de plusieurs instances de la tâche "vérifier transaction client". Le nombre de transactions étant variable d'un client à l'autre, le nombre de vérifications n'est connu qu'à l'exécution.

Démarrage déclenché par des signaux externes

Ces contrôles permettent à des signaux externes de démarrer l'exécution d'instances de tâches ou plus globalement de processus. Ils fournissent un moyen de coopération, à l'intérieur de l'entreprise ou avec des partenaires, notamment via plusieurs moteurs de flux de travail indépendants utilisés par différents départements. Par exemple, un processus d'évaluation des demandes de prêts défini sur un moteur de flux pourrait déclencher le processus d'octroi de prêts implémenté sur un autre moteur de flux.

- Déclencheur éphémère : Une instance de tâche peut être déclenchée par un signal provenant d'une autre partie du processus ou de l'environnement externe. Le déclencheur n'a d'effet que si une instance de la tâche concernée existe au moment de son émission. Ceci permet de lier des tâches à des événements ne devant être gérés qu'instantanément à leur réception.
- Déclencheur persistant : Ces déclencheurs sont gardés par le processus jusqu'à ce qu'une tâche puisse s'en charger. Ceci permet de lier des tâches à des événements devant obligatoirement être gérés.

4.2.3 Distribution des items de travail

Suite à leur création, les items de travail sont distribués sur les ressources pour être exécutés. L'affectation d'un item de travail à une ressource peut être initiée par le moteur de flux ou par la ressource elle-même. Un moteur de flux devrait aussi permettre de modifier les affectations de tâches aux ressources.

Distribution à la conception

Possibilité de délimiter, à la conception d'un item de travail, l'ensemble des ressources pouvant l'exécuter et les stratégies utilisées pour son affectation. La distribution à la conception permet de restreindre les autorisations des ressources et d'éviter une distribution inadéquate lors de l'exécution.

- Distribution directe : Spécifier à la conception l'identité de la/les ressource(s) auxquelles les instances d'une tâche sont distribuées à l'exécution.
- Distribution basée sur le rôle : Spécifier à la conception un ou plusieurs rôles ou compétences (agent de réception, expert comptable...) que possèdent les ressources auxquelles les instances d'une tâche peuvent être affectées.
- Distribution basée sur l'instance actuelle : Spécifier à la conception que les items de travail pour une instance de processus donnée sont allouées à une même ressource. Par exemple, toutes les tâches d'enregistrement d'une déclaration de sinistre sont effectuées par un même agent. L'agent en question sera déterminé lors de l'affectation de la première tâche du processus, à l'exécution.
- Distribution différée : Spécifier à la conception que l'identification des ressources auxquelles seront attribuées les instances de la tâche est différée jusqu'à l'exécution du processus. L'identité de la ressource peut être déterminée à l'exécution d'après un champ de données variable. La ressource sélectionnée peut être modifiée dynamiquement en mettant à jour la valeur de ce champ de données.
- Exécution automatique : Une instance de tâche est capable de s'exécuter sans utiliser les services d'une ressource. Cette possibilité est nécessaire pour gérer des tâches indépendantes de l'intervention de participants au flux de travail tel que l'archivage des opérations à la fin d'une journée de travail.

Allocation des ressources par le système de gestion du flux

Possibilité pour le système de gestion du flux d'offrir ou d'assigner de manière pro-active les items de travail nouvellement créés aux ressources. L'offre se fait en proposant les

items de travail à certaines ressources via une liste de travail partagée. L'allocation se fait en attribuant directement l'item à une ressource. Dans les deux cas, l'initiative est prise par le système de gestion du flux.

Cette fonctionnalité est essentielle au bon fonctionnement du flux de travail. Son évaluation se fait selon trois aspects :

- La manière de distribuer le travail : Le système peut offrir une tâche à une ou plusieurs ressources ou encore l'assigner à une ressource.
- Les stratégies de sélection de la ressource : Sélection aléatoire (non déterministe), sélection cyclique de l'ensemble des ressources éligibles (répartition équitable), sélection de la ressource qui a la file de travaux la plus courte (exécuter la tâche le plus tôt possible), etc.
- Le moment de distribution : offrir ou allouer des items de travail aux ressources avant leur activation (allouer la validation des déclarations de sinistre à l'employé x deux semaines avant qu'elles ne lui soient acheminées), lors de leur activation ou après leur activation (le traitement du sinistre x est alloué à l'employé y après l'acheminement du sinistre au bureau concerné). La distribution avant l'activation est utile pour les ressources capables de "réserver" des plages dans leur agenda et éviter ainsi de perdre du temps à allouer la tâche à la ressource durant l'exécution. La distribution des tâches lors de leur activation constitue l'approche la plus simple. La distribution après activation des tâches évite de surcharger les ressources avec des tâches en attente qu'elles n'exécuteront éventuellement pas.

Engagement des ressources à exécuter des tâches

Les ressources sont capables de s'engager, de leur propre initiative, à prendre en charge un item de travail. Les items leurs sont directement proposés par le système ou via une liste de travail partagée.

Cette capacité augmente l'autonomie des ressources. Son évaluation se fait selon trois aspects :

- Les cas d'engagement possibles : La ressource s'alloue un item pour l'exécuter plus

tard, la ressource initie l'exécution de l'item qui lui a été offert ou alloué. L'initiation de l'exécution par la ressource permet d'éliminer l'allocation des ressources par le système. Ce dernier propose alors les tâches à un large groupe de ressources concernées jusqu'à ce que l'une d'elle les revendique.

- La capacité pour une ressource de sélectionner une tâche selon ses propres préférences. Ceci lui donne le pouvoir et la flexibilité pour organiser elle même sa séquence de travail.

Modification des attributions de tâches aux ressources

Possibilité pour le système ou les ressources d'interrompre la distribution d'items de travail. La séquence des transitions d'états pour l'item de travail s'en trouve modifiée. La modification des attributions de tâches permet de parer à l'indisponibilité soudaine d'une ressource, mais aussi d'optimiser la distribution des tâches en prenant en compte des ressources nouvellement disponibles qui sont plus adéquates que celles présentement en charge de la tâche. Elle permet également aux ressources surchargées de s'alléger de tâches sous leurs responsabilités.

Ces opérations de modification, appelées détours dans les patrons de ressources (Russell et al., 2004b) englobent :

- La délégation : Une ressource alloue un item de travail qui lui est alloué à une autre ressource. Elle peut ainsi se décharger des tâches qu'elle n'est pas en mesure d'exécuter.
- L'escalade : Le système tente de faire progresser l'exécution d'un item de travail calé en l'offrant ou l'assignant à d'autres ressources.
- La désallocation : Une ressource rend disponible un item de travail qui lui est alloué pour une allocation ultérieure.
- La réallocation : Une ressource alloue un item de travail dont elle a initié l'exécution à une autre ressource.
- La suspension/reprise : Une ressource suspend temporairement l'exécution d'un item de travail ou reprend l'exécution d'un item précédemment suspendu. La suspension est utile lorsque la ressource atteint un point où elle ne peut plus avancer sur une

tâche donnée. Elle peut alors passer à une autre tâche et reprendre ultérieurement la tâche suspendue.

- Le saut : Une ressource décide d'ignorer l'exécution d'un item de travail qui lui est alloué. Ceci est utile lorsque l'item de travail n'est pas critique pour la finalisation du flux.
- La répétition (redo) : Une ressource répète l'exécution d'un item de travail complété précédemment. La répétition peut être nécessaire si l'exécution précédente ne s'est pas déroulée correctement ou si de nouvelles informations altérant le résultat de l'exécution sont désormais disponibles.

4.2.4 Représentation et utilisation des données dans le flux

Ces critères évaluent les différentes manières disponibles pour représenter et utiliser les données dans les flux de travail. Un premier critère jauge la diversité des contextes de définition de données. Les deux critères suivants analysent la diversité des types de transferts d'informations à l'intérieur des processus et avec l'environnement externe. La prise en charge de plusieurs mécanismes de transfert de données et l'utilisation des données comme préconditions et postconditions clôturent la section.

Définition de données pour un flux

Les données peuvent être représentées et utilisées dans différents contextes.

- Données locales aux instances d'une tâche : Peuvent stocker des résultats intermédiaires pendant l'exécution de la tâche.
- Données accessibles uniquement aux composants d'un bloc de tâches : Par exemple, des données sur les détails des valeurs mobilières ne sont visibles que pour les composants du bloc de tâches "évaluer les risques d'investissement" du processus "recommander investissement".
- Données spécifiques à une instance de processus : Données directement accessibles à tous les composants du processus pendant l'exécution de l'instance.
- Données accessibles à toutes les instances d'un flux : Par exemple, la grille des primes

de risque peut être utilisée par toutes les tâches de toutes les instances du processus "Rédiger une police d'assurance".

- Données d'environnement : Accéder à partir des processus à des données qui existent dans l'environnement d'exploitation externe. Cette fonctionnalité peut simplifier les processus et améliorer leur capacité à répondre aux changements et à communiquer avec des applications tierces tel que des bases de données ou des services externes.

Transfert de données entre composants d'un processus

Les données peuvent être transmises entre plusieurs types de composants d'un processus :

- Tâche à tâche : Communiquer des données entre deux tâches d'une même instance de processus. Lorsque les tâches s'exécutent chacune dans son espace d'adressage et qu'elles ne partagent pas de données globales significatives, les transferts de données entre tâches constituent un moyen d'échange d'informations.
- Envoi de données vers une tâche à instances multiples : Possibilité de distribuer sélectivement les données sur les instances de la tâche. Par exemple, la donnée "liste des nouvelles demandes d'emprunt" est passée aux instances de la tâche "traiter demande d'emprunt". Chaque instance reçoit une des demandes de la liste.
- Instance de processus à instance de processus : Passage de données entre deux instances de processus qui s'exécutent en parallèle. Ceci permet de communiquer des résultats obtenus durant l'exécution d'un processus à d'autres instances de processus.

Transfert de données entre processus et environnement

En plus de leur transmission à l'intérieur des processus, les données peuvent être échangées entre les processus et l'environnement externe. La réception de données provenant de l'environnement permet aux processus de s'informer sur l'état d'entités externes. L'envoi de données du processus à l'environnement permet, par exemple, de stocker les données de flux dans une base de données relationnelle.
On regroupe ces transferts en quatre catégories :

- Tâche - Environnement : Pouvoir passer des données d'une tâche à une ressource ou à un service de l'environnement d'exploitation et vice-versa.
- Instance de processus - Environnement : Par exemple, à la fin de l'exécution de chaque instance du processus "enregistrer nouveau client", les valeurs saisies sont transférées à une base de données pour y être stockées. Le transfert inverse permet d'informer chaque instance d'un processus des données environnementales les plus récentes.
- Processus - Environnement : Identique aux autres types de transferts à la différence que les données de processus sont globales à toutes les instances.

Prise en charge de divers mécanismes de transfert

La prise en charge de divers mécanismes de transfert de données permet de s'accorder aux contraintes d'espace d'adressage ou encore aux différences de formats de représentation des données. Les mécanismes de transfert de données comprennent :

- Passage par valeur : Permet d'avoir des composants écrits indépendamment les uns des autres. Les composants n'ont pas besoin de partager un espace de stockage de données commun ni de savoir quelles données leur sont passées. Le couplage est donc minimisé.
- Passage par référence : Utilise une référence de l'emplacement des données. L'emplacement doit être accessible par les participants à l'échange de données. L'avantage du transfert par référence est qu'il constitue un moyen implicite d'échange de données ne nécessitant pas de transporter les données entre les composants.
- Passage par copie : Possibilité pour une instance de processus de copier les valeurs d'un ensemble de données à partir d'une source externe dans son espace d'adressage au démarrage de son exécution. La valeur résultante à la fin de l'exécution est ensuite recopiée et renvoyée à la source. Le fait de manipuler des copies locales permet une indépendance par rapport aux changements de données et aux contraintes de concurrence.
- Transformation des données : Appliquer une fonction de transformation aux données avant qu'elles ne soient lues par un composant du processus ou après qu'il les ait

produites. Ceci permet de gérer les différences de représentation des données et de les accorder au format adéquat pour le contexte courant.

Utilisation des données comme conditions

Ces contrôles permettent de vérifier l'existence ou la validité de données pour démarrer ou compléter des tâches.

- Existence de données comme précondition ou postcondition d'une tâche : Une tâche ne peut commencer (précondition) ou être complétée (postcondition) que si un ensemble de données existe. Ceci permet de spécifier un traitement en cas d'absence de données et d'éviter ainsi de propager des erreurs d'exécution à l'ensemble des processus en exécution.
- Valeur de données comme précondition ou postcondition d'une tâche : Identique à la fonctionnalité précédente, à la différence que les données doivent non seulement exister, mais avoir des valeurs spécifiques pour qu'une tâche démarre ou soit complétée.

4.2.5 Accessibilité aux experts métier

Cette sous-section évalue les moyens fournis pour permettre aux utilisateurs métier d'administrer et d'utiliser les flux de travail.

Définition graphique des processus

Un utilisateur métier ne devrait pas définir les processus avec un langage informatique, aussi simple et expressif soit il, mais devrait pouvoir le faire via une notation graphique. En effet, les flux de travail représentent fondamentalement des diagrammes avec des structures de contrôle similaires à celles des diagrammes d'activité UML. De plus, la notation graphique BPMN s'est imposée comme un standard de fait, compréhensible par les utilisateurs métier.

Les outils de définition graphique raccourcissent également le temps de définition des processus. L'éditeur peut offrir davantage d'aide en permettant d'annuler, suspendre,

déléguer et réaffecter des instances de processus sans avoir à écrire du code.

Définition assistée des activités

En assistant les utilisateurs métier dans la définition des activités, on abstrait les détails de programmation.
L'assistance peut se manifester sous plusieurs formes, tel que :

- Des outils de génération de formulaires qui sont intégrés aux outils de définition des processus.
- La possibilité de définir la partie exécution d'une activité en combinant des éléments d'une bibliothèque d'actions basiques, soit via un outil visuel soit via un langage de script.
- Des outils facilitant l'intégration de systèmes bureautiques, de manipulation d'images ou de gestion de documents.

Interfaces pour non informaticiens

Il s'agit de fournir des interfaces destinées aux non informaticiens. Idéalement, l'utilisateur peut y accéder à partir d'un navigateur web.
À partir de ces interfaces, il devrait être possible de gérer la liste de travail, exécuter des activités, représenter graphiquement l'état d'un processus ou encore démarrer une instance de processus.
Ces fonctionnalités devraient être contrôlées en imposant des contraintes aux utilisateurs. Ceci permet de ne pas les submerger avec trop d'options et assure une édition sans danger des flux.

Abstraction des détails d'implémentation

Les utilisateurs métier doivent être capables de définir et utiliser des flux de travail sans manipuler le langage d'exécution des flux. Les processus qu'ils modélisent graphiquement doivent être automatiquement implémentés dans le langage du moteur de flux de

travail, de manière invisible à l'utilisateur.

Modèles de processus

Les modèles de processus, à l'instar des modèles de règles, permettent de réutiliser des structures de flux de travail récurrentes. Les modèles définissent une séquence d'activités, de branchements et de participants pour des cas d'utilisation fréquents. Dans le domaine de la finance, on envisagerait des modèles simples pour la réalisation de transactions de débits par exemple, ou des modèles plus complexes pour l'approbation d'une demande de prêt ou encore pour la gestion des déclarations de sinistre chez un assureur.
Un système de gestion de flux de travail qui offre des modèles prédéfinis et qui permet de les modifier ou d'en créer de nouveaux factorise les structures récurrentes. Au fur et à mesure qu'un organisme utilise le produit, il agrandit sa bibliothèque de modèles personnalisés et les perfectionne. Ceci permet un gain de temps et une uniformisation des processus à travers l'organisme.

4.2.6 Le moteur de flux de travail

Les critères suivants se rapportent à des caractéristiques propres au moteur d'exécution des flux de travail.

Intégration à l'environnement

À l'image du moteur de règles, le moteur de flux de travail doit s'intégrer à une solution informatique globale. Pour cela, il doit être capable de communiquer avec les autres composants du système. L'intégration se fait à plusieurs niveaux. À la couche présentation, il serait pratique de pouvoir utiliser plusieurs types d'interfaces, en fonction des systèmes informatiques des organismes clients. Au niveau de la couche métier, un moteur de flux intégrable aux serveurs d'applications ou aux fils de messages pourra facilement interagir avec des modules externes. Enfin, au niveau couche de données, un moteur capable d'importer et d'exporter des informations de sources externes échangera

facilement les données avec les autres composants du système.

Plusieurs indicateurs permettent de mesurer la capacité d'intégration d'un moteur de flux de travail, notamment :

- Les plateformes et systèmes d'exploitation pris en charge.
- Les étapes nécessaires à l'intégration.
- Le niveau d'automatisation des communications du moteur avec l'ensemble du système.
- L'existence d'APIs exhaustives et bien conçues pour l'interfaçage avec le moteur de flux.
- La possibilité d'utiliser différentes implémentations de la couche de présentation.
- La possibilité d'utiliser des services web ou d'autres applications tierces comme participants au flux.
- La possibilité d'invoquer des applications tierces pour réaliser des tâches.
- L'existence d'intégrations prédéfinies avec des versions récentes de serveurs d'applications.

Changements dynamiques

Capacité de changer les définitions des procédures à chaque moment pendant que le serveur est en exécution : les changements peuvent être pour une instance unique d'un processus ou pour toutes les instances qui n'ont pas encore atteint l'activité à modifier. Cette fonctionnalité permet de traiter les situations imprévues et de s'adapter au changements. La capacité de déployer de nouvelles versions d'une définition de processus sans pouvoir migrer les instances déjà démarrées ne satisfait que partiellement le critère.

4.2.7 Outils de vérification

La vérification de la syntaxe, le test, le débogage, la journalisation et l'audit constituent des outils de vérification utiles pour le développement et la maintenance de flux de travail.

Vérification de la syntaxe

Si le moteur de règles utilise un langage structuré pour la définition des flux tel que XPDL ou BPEL, il est utile qu'un vérificateur syntaxique surligne les mots clés, les variables et les valeurs avec des couleurs différentes.
La vérification de la syntaxe permet de s'assurer en temps réel de la conformité au format du langage. Le surlignage permet de mieux visualiser la structure globale du code.

Test

Les tests unitaires et les suites de tests permettent d'exécuter du code et de vérifier que le résultat correspond au comportement requis. Les tests permettent de détecter les erreurs avant que les utilisateurs ne le fassent et de les réparer rapidement et à un faible coût.
Pour valider ce critère, un système de gestion de flux de travail doit disposer d'une prise en charge incorporée des tests et des simulations sans utiliser d'outils externes. Il doit aussi permettre la spécification et l'exécution de suites de tests. La possibilité d'offrir des simulations animées des processus où la séquence d'activation des tâches et les différentes interactions sont représentées sur le modèle graphique du flux est un atout non négligeable.

Débogage

Le débogage dans le contexte des moteurs de flux de travail signifie l'inspection détaillée de l'exécution d'une instance de flux. Ceci est réalisé notamment avec l'inspection et les points d'arrêt.
Les points d'arrêt suspendent l'exécution d'une instance de processus à un certain stade. Le traitement peut être repris à la demande de l'utilisateur.
Les gardes ou *watchpoints* sont utilisés pour suivre de près l'état des instances de processus, des tâches et des données qui leur sont associées. Le traitement n'est pas suspendu

comme c'est le cas pour les points d'arrêt. Les états des objets et les événements sont enregistrés au fur et à mesure.

Ce critère est satisfait s'il est possible de :

- Définir des points d'arrêt et des gardes pour les instances de processus et de tâches ainsi que pour les données qu'ils manipulent.
- Observer des informations en différé, sans stopper l'exécution du programme.
- Visualiser les étapes d'exécution par lesquelles passe le moteur de flux et les parcourir en affichant leurs opérations internes.

Journalisation et audit

La journalisation permet de capter l'information à propos des opérations du moteur de flux. Une fois enregistrées, ces données peuvent être utilisées à des fins d'audit. L'audit permet de savoir comment, pourquoi et quand une décision donnée a été prise. Il permet également de dégager des statistiques sur différents aspects de l'édition et de l'exécution des flux.

Une journalisation de qualité devrait permettre :

- La journalisation hiérarchique par granularité des événements.
- La catégorisation par niveaux (niveau erreur, avertissement, information...) afin de contrôler le niveau de détails des événements à journaliser.
- La génération de fichiers de journalisation sous formats multiples (texte plat, XML, HTML...).

Pour ce qui est de l'audit, un système de gestion de flux de travail doit présenter les informations journalisées de sorte que l'utilisateur puisse facilement les parcourir et les filtrer afin d'extraire les données dont il a besoin.

4.2.8 Stockage des processus

Cette sous-section comprend des critères relatifs au stockage et à l'exportation des définitions de processus.

Versionnage

Il s'agit d'affecter des noms ou des numéros de versions uniques à des états uniques des définitions de processus.
Le versionnage permet de comparer des définitions de processus avec des révisions plus anciennes ou plus récentes et d'en dégager les différences. Le versionnage peut s'appuyer sur un système collaboratif de contrôle de version.
Un système implémentant le versionnage des processus est capable de :
- Versionner les fichiers de définition des processus et contrôler leur accès.
- Versionner des projets entiers afin de gérer des versions de publication du système.
- Revenir à une version précédente (rollback) et exécuter le processus tel qu'il était à un instant passé.

Importation/Exportation de processus

La possibilité d'importer et d'exporter des processus dans divers formats améliore la coopération entre moteurs de flux de travail et permet de se libérer des formats spécifiques à un produit particulier. La possibilité d'importer et d'exporter selon les formats standards tels que BPMN, XPDL et BPEL est un atout important dans l'évaluation de ce critère.

4.3 Critères non fonctionnels

Les aspects globaux relatifs à l'exploitation des moteurs de flux sont similaires à ceux des moteurs de règles avec l'ajout du critère de conformité aux standards.

4.3.1 Conformité aux standards

Le modélisation graphique des flux de travail possède un standard de fait qui est BPMN. De même, l'échange de définitions de processus se fait essentiellement sous format XPDL ou BPEL. Ainsi, contrairement aux moteurs de règles, les moteurs de flux disposent

de standards établis et largement utilisés dans l'environnement d'entreprise. Les solutions à évaluer conformes aux standards permettent une meilleure coopération et sont indépendantes d'un quelconque format propriétaire.

4.3.2 Performance et coûts en ressources

Performance des moteurs de flux

Contrairement aux moteurs de règles, les moteurs de flux n'exécutent pas d'opérations de calcul volumineuses mais sont concernés par la distribution des tâches aux acteurs et le contrôle de la séquence d'exécution. Les opérations nécessitant une capacité de traitement élevée sont principalement comprises dans les tâches et ne rentrent donc pas dans les responsabilités du moteur de flux de travail. C'est pour cela, qu'à notre connaissance, il n'existe pas beaucoup de publications concernant l'évaluation des performances des moteurs de flux.

Cependant, certains travaux s'intéressent à cet aspect. Gilmann, Vildermann et Veikum (Gillmann, Mindermann et Weikum, 2000) relèvent que les bancs d'essais devrait jauger l'efficacité de l'implémentation des diverses constructions de contrôle de flux proposées par le moteur tel que la divergence conditionnelle, la divergence parallèle, les structures d'itération...

Par ailleurs, pour les flux comprenant des tâches interactives, la performance du moteur de flux est mesurée pour les interactions avec l'utilisateur au démarrage et à la complétion des tâches. Ces interactions comprennent l'allocation des tâches, le transfert de données, la manipulation de la liste de travail des ressources...

Coût en ressources matérielles

La solution proposée par la Chaire se destine potentiellement à des organismes disposant de ressources matérielles limitées. Il est donc important que les produits utilisés aient des exigences matérielles raisonnables. La consommation en mémoire vive, en espace de stockage et en temps de calcul sont autant de facteurs à considérer dans l'évaluation du

coût en ressources.

4.3.3 Prise en main

Comme pour les moteurs de règles, la facilité de prise en main englobe :

- La qualité et l'accessibilité de la documentation.
- La convivialité du produit : temps requis pour l'installation, facilité de l'installation, convivialité des interfaces utilisateur.
- La courbe d'apprentissage : temps nécessaire pour se familiariser avec le produit et en faire une utilisation optimale.

4.3.4 Évaluation sur l'aspect logiciel libre

Comme pour les moteurs de règles (voir Section 3.2.3), l'évaluation sur l'aspect logiciel libre des systèmes de gestion de flux comprend :

- La communauté : taille, niveau d'activité et niveau de compétence de la communauté.
- La fréquence de publication : indique le niveau d'implication des développeurs.
- La longévité : période d'existence du projet ; indique son niveau de maturité.
- La licence : clauses de la licence en accord avec l'utilisation visée par la Chaire.
- L'assistance communautaire : existence d'une assistance usager et d'une assistance traitant les défaillances.
- L'existence d'une assistance professionnelle : service d'assistance professionnel payant, signe que le produit jouit d'une utilisation conséquente en entreprise.

4.4 Pondération des critères d'évaluation selon l'environnement

Comme pour les moteurs de règles, on met en contexte les critères d'évaluation des moteurs de flux de travail en définissant des scénarios d'utilisation qui illustrent des prototypes d'environnements d'entreprises.

Les scénarios expriment des orientations globales et peuvent être modifiés pour représenter au mieux les caractéristiques de chaque organisme. Ils peuvent également être combinés,

à l'exception des scénarios 3 et 4 qui sont mutuellement exclusifs. Si plusieurs scénarios sont applicables à un organisme, le poids le plus important attribué pour chaque critère devrait être pris.

4.4.1 Critères invariablement importants

Lors de l'évaluation des systèmes de gestion du flux de travail, certains critères sont importants quelque soit l'environnement de l'organisme client :

- Le langage de flux doit comprendre le modèle de données.
- Les contrôles élémentaires du flux doivent nécessairement être réalisables.
- La prise en charge des contrôles avancés du flux est toujours un atout.
- Le système de gestion du flux doit être capable d'allouer les tâches aux ressources.
- Les fonctions de journalisation et d'audit sont nécessaires pour garder une trace des exécutions.
- Les aspects liés à la nature libre de la solution sont toujours pertinents, particulièrement la conformité de la licence aux utilisations visées, la fréquence de publication et la longévité du produit.

4.4.2 Scénario 1 : Collaboration conséquente entre le moteur de flux et l'environnement externe

Dans ce scénario, le flux de travail implique l'utilisation de plusieurs applications tierces utilisées dans l'entreprise (client de messagerie, application de traitement de texte...). Plusieurs informations sont échangées avec l'environnement et les processus exportent certains résultats vers une base de données relationnelle pour y être stockés. Certains processus nécessitent l'échange avec des organismes partenaires.

L'emphase est donc mise sur les aptitudes de coopération dont dispose le moteur de flux. Dans ce cadre, les critères suivants sont particulièrement importants :

- Prise en charge de plusieurs moyens d'expression des processus.
- Démarrage de tâches déclenchées par des signaux externes.
- Transfert de données entre le processus et l'environnement externe.

- Prise en charge de divers mécanismes de transfert de données.
- Intégration à l'environnement afin de communiquer avec les autres composants du système.
- Importation/Exportation de processus.
- Conformité aux standards de modélisation (BPMN) et de langage d'exécution et d'échange des flux (XPDL / BPEL).

4.4.3 Scénario 2 : Agilité, besoin d'adaptation au changement et d'autonomie des composants

Ce second scénario représente un contexte agile et évolutif. Les processus changent fréquemment, la disponibilité des ressources varie beaucoup et plusieurs imprévus peuvent survenir. L'organisme possède des succursales localisées dans des environnements différents. Les processus sont variés et différents. Les participants au flux sont hiérarchisés selon plusieurs départements et rôles distincts. Ils doivent être autonomes afin de faire face au contexte agile.

Ce scénario met l'accent sur l'adaptation au changement, la gestion de contextes divers et l'autonomie des ressources.

Les critères pertinents pour l'adaptation au changement sont :

- La prise en charge de plusieurs moyens d'expression des processus.
- L'annulation d'éléments dans le flux afin ne pas exécuter des opérations invalidées.
- La modification des attributions de tâches aux ressources pour parer à l'indisponibilité soudaine d'une ressource.
- La prise en charge des changements dynamiques afin de traiter rapidement les changements.
- Le versionnage des définitions de processus.

Pour ce qui est de la diversité des contextes, les critères importants seraient :

- La prise en charge de l'internationalisation.
- La définition des données pour un flux selon plusieurs contextes afin d'en contrôler la visibilité.

- La distribution à la conception pour restreindre les droits des ressources à exécuter des tâches.

Enfin, l'autonomie est réalisée en permettant l'engagement des ressources à exécuter des tâches.

4.4.4 Scénario 3 : Organisme de moyenne ou grande taille avec des flux complexes et un département informatique interne

Même si les effectifs et l'infrastructure dont disposent les corporations financières capitalistes sont généralement plus importants que ce qui existe dans les entreprises de finance sociale, les flux de travail n'y sont pas forcément plus complexes. En effet, le portefeuille de produits offert en finance sociale est large et varié. De plus, certains organismes de finance sociale disposent d'une infrastructure importante. Dans ce troisième scénario, l'organisme utilise des processus complexes impliquant des contrôles de flux sophistiqués et une exploitation avancée des données. Le département informatique interne est important. Des informaticiens sont employés à temps plein et sont disposés à prendre en charge le système de gestion des flux.

Ces organismes de l'économie sociale de moyenne ou grande taille doivent disposer de moteurs de flux de travail capables de réaliser des contrôles de flux sophistiqués, de permettre une exploitation avancée des données, de factoriser des modèles de processus, de versionner les définitions de processus et d'offrir des outils de vérification syntaxique.

Les critères importants pour ce scénario sont donc :

- La prise en charge de plusieurs moyens d'expression des processus.
- Le contrôle selon l'état du processus.
- L'itération dans le flux pour saisir des comportements répétitifs.
- La gestion d'instances multiples d'une même tâche dans une instance de processus.
- L'utilisation des données comme préconditions et postconditions d'exécution des tâches.
- Le transfert de données entre composants d'un processus.
- Les performances du moteur de flux.
- La possibilité de définir des modèles de processus.

- Le versionnage des définitions de processus.
- La vérification de la syntaxe du code d'exécution car les développeurs auront probablement à manipuler directement du code exécutable sous format XPDL ou BPEL par exemple.
- L'existence d'outils de test et de débogage des flux à disposition du département informatique.

4.4.5 Scénario 4 : Organisme limité en ressources logistiques et en personnel informatique

Ce quatrième scénario concerne les organismes de finance sociale et solidaire de petite taille. Typiquement, ce type d'organismes ne dispose pas d'un département informatique interne conséquent et les ressources matérielles y sont limitées. Le personnel non technique est enthousiaste pour s'investir dans la création des processus, avec l'aide des collègues informaticiens. Les directeurs ne sont pas forcément compétents en programmation mais comprennent bien les flux de travail internes à l'entreprise.
Pour ce type d'organismes, certains critères sont particulièrement importants :

- L'ensemble des critères relatifs à l'accessibilité aux experts métier.
- La possibilité de définir des modèles de processus.
- De faibles coûts en ressources matérielles.
- Une documentation de qualité, une interface conviviale et une faible courbe d'apprentissage.
- Une communauté et une assistance de qualité.
- L'existence d'une assistance professionnelle.

4.5 Grille d'évaluation des moteurs de flux de travail

L'ordre des colonnes des scénarios S1, S2, S3 et S4 correspond à leur ordre de définition à la section précédente :

- S1 : Collaboration conséquente entre le moteur de flux et l'environnement externe.
- S2 : Agilité, besoin d'adaptation au changement et d'autonomie des composants.

- S3 : Organisme important avec des flux complexes et un département informatique interne.
- S4 : Organisme limité en ressources logistiques et en personnel informatique.

Attribut	S1	S2	S3	S4
A. Critères fonctionnels				
Langage de définition des processus				
1. Prise en charge de plusieurs moyens d'expression	4	4	4	3
2. Prise en charge de l'internationalisation	4	5	3	3
3. Compréhension du modèle de données	5	5	5	5
Contrôle du flux de travail				
4. Contrôle élémentaire du flux	5	5	5	5
5. Contrôle avancé du flux	5	5	5	5
6. Contrôle selon l'état du processus	4	5	5	3
7. Itération dans le flux	4	4	5	4
8. Annulation d'éléments dans le flux	3	5	4	3
9. Gestion d'instances multiples	3	3	5	3
10. Démarrage déclenché par des signaux externes	5	4	4	3
Distribution des items de travail				
11. Distribution à la conception	3	5	4	3
12. Allocation des ressources par le système de gestion du flux	5	5	5	5
13. Engagement des ressources à exécuter des tâches	4	5	3	3
14. Modification des attributions de tâches aux ressources	3	5	4	3
Représentation et utilisation des données dans le flux				
15. Définition des données pour un flux	4	5	4	3
16. Transfert de données entre composants d'un processus	4	4	5	3
17. Transfert de données entre processus et environnement	5	3	4	3
18. Prise en charge de divers mécanismes de transfert	5	4	3	3
19. Utilisation des données comme conditions	3	3	4	3

Accessibilité aux experts métier				
20. Définition graphique des processus	4	4	3	5
21. Définition assistée des activités	4	4	3	5
22. Interfaces pour non informaticiens	4	4	3	5
23. Abstraction des détails d'implémentation	4	4	3	5
24. Modèles de processus	4	4	5	5
Le moteur de flux de travail				
25. Intégration à l'environnement	5	4	4	4
26. Changements dynamiques	3	5	4	3
Outils de vérification				
27. Vérification de la syntaxe	4	4	5	3
28. Test (Simulation)	3	3	5	3
29. Débogage	3	3	5	3
30. Journalisation et audit	4	4	4	4
Stockage des processus				
31. Versionnage	3	5	4	3
32. Importation/Exportation de processus	5	4	3	3
B. Critères non fonctionnels				
33. Conformité aux standards	5	4	4	4
Performance et coût				
34. Performance des moteurs de flux	3	3	4	3
35. Coûts en ressources matérielles	3	3	3	5
Prise en main				
36. Documentation	3	3	3	5
37. Convivialité	3	3	3	5
38. Courbe d'apprentissage	4	4	4	5
Évaluation sur l'aspect logiciel libre				
39. Communauté	4	4	4	5

40. Publications	4	4	4	4
41. Longévité	4	4	4	4
42. Licence	5	5	5	5
43. Assistance communautaire	4	4	3	5
44. Assistance professionnelle	3	3	2	5

Tableau 4.2: Grille d'évaluation des moteurs de flux

Souvent, une partie des facteurs menant à l'adoption d'un produit au sein d'une entreprise sont subjectifs. Il existe clairement une dose de subjectivité dans toute évaluation globale comme celles que nous faisons des moteurs de règles et des moteurs de flux de travail. Cependant, nous avons essayé de minimiser ses effets. Avec l'utilisation d'autant de facteurs d'évaluation précis, il est effectivement difficile de biaiser les résultats. Les grilles conçues devraient donc produire des résultats fiables et pertinents. C'est donc munis de nos grilles d'évaluations exhaustives que nous attaquons l'évaluation des moteurs de règles et des moteurs de flux de travail.

CHAPITRE V

ÉVALUATION DES MOTEURS DE RÈGLES

Maintenant que nous avons défini nos propres grilles d'évaluation des moteurs de règles et de flux de travail, nous allons sélectionner les produits à évaluer et les évaluer selon ces grilles.

Par rapport à un critère, un produit se verra attribuer une des trois notes suivantes :

- 0 : le produit ne satisfait pas le critère ou le respect du critère est très faible.
- 1 : le produit répond partiellement au critère. Pour les critères fonctionnels, la fonctionnalité est réalisable indirectement, ou demande des efforts de développement considérables. Pour les critères non fonctionnels, l'aspect évalué est partiellement satisfait.
- 2 : le produit répond au critère (la fonctionnalité est directement prise en charge, l'aspect non fonctionnel est suffisamment satisfait...)

Ce système de notation est complémentaire avec la mise en contexte selon des environnements types. En effet, le score d'un produit n'augmentera pas s'il ne prend pas en charge un critère donné et il augmentera proportionnellement au poids du critère et au niveau de satisfaction (poids * 1 ou poids * 2).

5.1 Présentation des produits à évaluer

Afin de délimiter l'évaluation des systèmes de gestion de règles existants, nous allons effectuer une présélection des trois systèmes les plus pertinents. Parmi le large éventail

de produits à code ouvert, nous choisirons les trois les plus activement développés, les plus aboutis et les plus susceptibles d'être utilisés dans un environnement de production. Au rayon des moteurs de règles écartés de l'étude, nous citons :

- Hammurapi Rules [1] est un moteur d'inférence qui offre aussi bien le chaînage avant que le chaînage arrière. L'écriture des règles se fait en Java, ce qui restreint le contrôle des utilisateurs qui n'ont pas de compétences de programmation. Cependant, les règles sont paramétrables et flexibles, ce qui permet de les modifier sans toucher au code. Les ensembles de règles sont assemblés et configurés avec XML. Le produit est activement développé. Cependant, le produit se limite au moteur d'inférence, et ne constitue pas un système de gestion des règles complet.
- JRuleEngine [2] est un moteur de règle conforme à la spécification Java JSR-94. Comparé à d'autres produits conformes à la spécification, JRuleEngine est minimaliste (la taille totale du produit à télécharger est de 280kb). De plus la dernière version publiée remonte à avril 2008.
- Zilonis RuleEngine [3] est un moteur de règles libre qui est encore en version beta (0.97b), ce qui indique qu'il n'est pas encore prêt pour une utilisation dans un environnement de production. D'autre part, le produit ne semble pas être activement développé puisque les dernières validations dans le dépôt SVN du produit remontent à 2 ans.

Les trois produits présélectionnés sont le système Drools de JBoss, la solution OpenRules et le système CLIPS. Les trois produits sont matures, proposent un ensemble conséquent de fonctionnalités et ont été largement utilisés dans des environnements de production réels.

---

1. http ://www.hammurapi.biz (dernière consultation le 09-11-10)
2. http ://jruleengine.sourceforge.net/ (dernière consultation le 09-11-10)
3. http ://www.zilonis.org/ (dernière consultation le 09-11-10)

5.1.1 Drools [4]

Drools est un système de gestion de règles développé par JBoss, filiale du groupe RedHat. Le moteur est une implémentation améliorée des moteurs de règles appelée ReteOO, qui est basé sur une version modifiée de l'algorithme Rete pour la machine virtuelle Java. Drools permet la définition des règles en programmation déclarative (qui permet de dire Que faire et non pas Comment faire). Les DSLs (Domain Specific Languages) sont pris en charge, ce qui permet d'exprimer les règles selon la sémantique spécifique au domaine, avec un langage proche du langage naturel. Drools fournit des outils d'édition graphique (tables de décision dans des tableurs, métaphores visuelles), des outils web et des outils de productivité pour développeurs. Il existe également un plugiciel *eclipse* avec auto-complétion, fenêtres de débogage et interfaces graphiques représentant les flux de règles. La dernière version Drools 5 apporte plusieurs améliorations dont :

- Drools Flow qui permet de définir des flux de règles.
- Drools Fusion qui permet une gestion avancée des événements : hiérarchies d'événements, corrélation d'événements, patrons complexes pour un ensemble d'événements...

Drools est disponible sous la licence libre Apache 2.0.

5.1.2 CLIPS [5]

CLIPS (C Language Integrated Production System) est un environnement pour la construction de systèmes experts basés sur des règles et/ou des objets. Crée par la NASA en 1985, il est maintenant largement utilisé par les gouvernements, l'industrie et les universités. Il est encore activement développé. CLIPS prend en charge les paradigmes de programmation procédurale, orientée objet ou basée sur les règles. Il est écrit en C et peut être installé sur n'importe quel système possédant un compilateur C ou C++ répondant à la norme ANSI.

---

4. http ://www.jboss.org/drools (dernière consultation le 09-11-10)

5. http ://CLIPSrules.sourceforge.net (dernière consultation le 09-11-10)

CLIPS peut être appelé comme sous-routine ou intégré dans des applications codées en C, Java ou Python, entre autres. Il peut être facilement étendu à travers l'utilisation de plusieurs protocoles prédéfinis. Le produit fournit un environnement de développement qui inclut des aides au débogage et un éditeur intégré. Des environnements plus riches ont été développés pour les systèmes Unix, Mac et Windows.
Enfin, CLIPS est maintenu comme logiciel du domaine public. Il est donc permis de l'utiliser, de le modifier et de le redistribuer gratuitement et à volonté.

5.1.3 OpenRules [6]

OpenRules est un système de gestion de règles complet qui fournit une suite d'outils à code ouvert pour le développement d'applications basées sur les règles d'affaires. Il s'interface avec les tableurs MS Excel et Google Speadsheets, fournit un plugiciel *eclipse* et un dépôt de règles d'affaires. Les développeurs peuvent utiliser OpenRules à partir des langages Java et .NET. Le projet propose une suite complète de composants couvrant un large spectre dans le domaine des règles d'affaires. La suite inclut l'exécution de règles (Rules Engine), leur gestion (Rules Repository), leur création avec des tables décisionnelles (Rules Editors), leur test et validation (Rules Validator et Rules Testing), la génération d'interfaces graphiques (Rules Forms) ainsi que des APIs Java, des services web et des applications JSP (Rules Deployment). OpenRules est disponible sous licence GPLv2 pour les projets à code ouvert et non GPL pour les projets à code fermé. La version la plus récente (5.3.2 à l'écriture du mémoire) est payante. En effet, un logiciel libre ne signifie pas forcément qu'il est gratuit. Toutefois, la version antérieure (4.1), qui a été utilisée dans des environnements de production réels chez plusieurs corporations importantes [7], est gratuite et également sous licence GPL. On évaluera cette version 4.1 pour éviter d'imposer des charges financières supplémentaires aux organismes de finance solidaire et sociale.

---

6. http ://openrules.com/ (dernière consultation le 09-11-10)

7. http ://openrules.com/whatTheySay.htm (dernière consultation le 09-11-10)

5.2 Langage de définition des règles

5.2.1 Prise en charge de plusieurs moyens d'expression des règles

Drools

Drools permet d'exprimer les règles avec :

- le langage de règles par défaut DRL combiné au choix à du code Java ou MVEL. MVEL [8] est un langage de script pour Java qui permet d'accéder aux objets imbriqués, aux collections et aux tableaux de manière élégante. Par exemple, le code Java

  ```
  $personne.getAddresses().get("maison").setNomRue("ma rue");
  ```

  pourrait être écrit en MVEL de la manière suivante

  ```
  $personne.addresses["maison"].nomRue = "ma rue";
  ```

- un langage spécifique au domaine d'affaires (DSL) défini dans des fichiers .dsl et utilisable dans des fichiers .dslr. Le langage DSL est à définir par l'utilisateur et peut être enrichi au fur et à mesure. Chaque entrée dans le fichier .dsl définit une expression selon le langage DSL et son équivalent dans le langage DRL. Les paramètres sont déclarés entre accolades.
- des tables décisionnelles sous format de tableur (Excel, OpenOffice, Google Spreadsheet...) ou de fichier CSV (Comma-Separated Values).
- le langage de systèmes experts CLIPS via l'extension officielle drools-CLIPS.
- le langage Scala via un projet tierce nommé drools-scala [9].
- le langage Groovy via drools-groovy, qui semble être un projet délaissé [10].

Ainsi, de la syntaxe formelle DRL au langage naturel possiblement défini dans un DSL, en passant par les langages dynamiques tel que Groovy ou fonctionnels tel que CLIPS

---

8. http ://mvel.codehaus.org/ (dernière consultation le 09-11-10)

9. http ://wiki.github.com/memelet/drools-scala/ (dernière consultation le 09-11-10)

10. http ://legacy.drools.codehaus.org/Groovy+Semantic+Module (dernière consultation le 09-11-10)

et Scala ou encore les tables décisionnelles, le système de gestion de règles Drools offre une panoplie d'approches très diversifiées pour exprimer les règles. Un utilisateur métier pourrait être à l'aise avec les tables décisionnelles et le langage DSL. Les développeurs ayant précédemment utilisé CLIPS ou étant plus à l'aise avec une approche fonctionnelle pourrait mettre à profit leur expertise avec drools-CLIPS et drools-scala. Enfin, les adeptes de langages dynamiques peuvent utiliser Groovy pour concevoir les règles. Drools obtient la note de 2 sur 2 pour ce critère.

CLIPS

Les règles sont exprimées dans CLIPS exclusivement avec la commande *defrule*.

```
(defrule nom-regle
(conditions)
=>
(actions declenchees si les conditions sont satisfaites)
)
```

Les conditions de déclenchement d'une règle et les actions à exécuter peuvent être exprimées :

- avec un langage procédural à la LISP : Les faits sont définis comme structures avec les commandes *deftemplate* (structure d'un enregistrement) et *deffacts* (ensemble d'assertions). Les actions sont exprimées avec des fonctions (*deffunction*) et des commandes CLIPS (assertion/rétraction de faits, affichage...).
- avec un langage orienté objet nommé COOL (CLIPS Object Oriented Language) : Les faits peuvent être des instances de classes et les actions peuvent être des appels à des méthodes sur ces instances. Le modèle objet de CLIPS prend en charge l'abstraction (classes abstraites et méthodes génériques), l'héritage simple et multiple (sous-classes directes et indirectes), l'encapsulation (accesseurs et mutateurs automatiquement générés par CLIPS pour chaque attribut d'une classe), le polymorphisme (appel de la méthode adéquate selon le type du receveur) et la liaison dynamique (surcharge de fonctions).

Ainsi, au sein du langage CLIPS, les développeurs ont le choix entre une approche procédurale, une approche orientée objet ou un mélange des deux approches selon leurs aptitudes et leurs préférences. En dehors du langage natif :

- Le langage C, et par extension C++ sont capables de s'interfacer avec CLIPS via l'API originale. D'autres langages tel que ADA, Perl (CAPE), Python (PyCLIPS) ou Java (CLIPSJNI) peuvent le faire via des bibliothèques fournies par la communauté. En fonction du niveau d'intégration avec CLIPS fourni par ces bibliothèques, on est capable d'invoquer le moteur de règles à partir du langage tiers ou d'appeler des méthodes définies avec ce langage dans les règles CLIPS.
- Plusieurs variantes enrichissent les fonctionnalités du système et rajoutent des constructions et des paradigmes de programmation nouveaux pour exprimer les règles : ajout de la logique floue avec fuzzyCLIPS, d'agents intelligents avec dynaCLIPS, d'agents CORBA avec CLIPS Corba Network Protocol...

On note malgré tout que les langages de programmation tiers pouvant s'interfacer avec CLIPS ne sont pas directement pris en charge et doivent invoquer le système de règles comme un service externe. De plus, CLIPS ne propose pas une prise en charge directe des tables décisionnelles et des fichiers CSV. Le produit obtient donc la note de 1 sur 2 pour ce critère étant donné qu'il propose deux approches de programmation au sein d'un langage unique ainsi que des extensions et des points d'intégration avec d'autres langages de programmation, mais que la définition de règles ne se fait qu'avec une construction *defrule* du langage CLIPS.

OpenRules

La particularité d'OpenRules est qu'il vise à exprimer les faits, les règles et les interfaces utilisateur exclusivement avec des tableurs. Les instructions d'évaluation des conditions et d'exécution des actions des règles sont écrites en Java. La bibliothèque Apache POI permet la manipulation des tableurs à partir de code Java et des bibliothèques OpenRules font le moulinage du contenu des tables décisionnelles et établissent le lien entre ces données, les méthodes Java et les faits. La version 5.3, qui est payante pour une uti-

lisation commerciale, permet de définir les règles en dehors des tableurs. Les règles de salutation exprimées dans la table décisionnelle de la figure 5.1 peuvent être exprimées selon les formats suivants :

Rules void **definirSalutation**(int hour, Reponse reponse)		
De	A	Salutation
0	11	Bon Matin
12	17	Bonjour
18	22	Bonsoir
23	24	Bonne Nuit

Figure 5.1 Table décisionnelle de règles de salutation

- Fichiers XML : Le document n'utilise pas de spécification particulière pour valider sa structure. Le développeur doit faire le lien entre les balises et leur signification dans le cadre d'OpenRules.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReglesSalutation ruleTableName="reglesSalutation" templateName="
    definirSalutation" type="Rangee de regle(s)">
  <Regle de="0" a="11" salutation="Bon Matin" />
  <Regle de="12" a="16" salutation="Bonjour" />
  <Regle de="17" a="21" salutation="Bonsoir" />
  <Regle de="22" a="24" salutation="Bonne Nuit" />
</ReglesSalutation>
```

- Code Java : Simples tableaux de chaînes de caractères reprenant la structure d'une table décisionnelle. Ici encore, c'est au développeur d'établir le lien entre les éléments du tableau et leur signification dans le cadre d'OpenRules.

```
reglesExternes.addRuleTable(
        "SalutationExterne", //nom de la table de regles
        new String[][] {          //regles
                new String[] {"0","11","Bon Matin"},
                new String[] {"12","17","Bonjour"},
                new String[] {"18","21","Bonsoir"},
                new String[] {"22","24","Bonne Nuit"}
        }
);
```

- Table dans une base de données : Les champs de la table représentent des colonnes et les enregistrements des lignes d'une table décisionnelle.

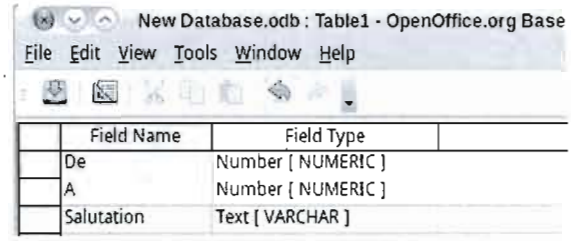


Figure 5.2 Table d'une BD relationnelle des règles de salutation

Il est clair que, même si les supports changent, l'approche est exactement la même. Il s'agit simplement de la "simulation" de tables décisionnelles en utilisant Java, une base de données ou un fichier XML. On ne peut donc pas considérer OpenRules comme offrant plusieurs moyens d'expression des règles. Le système cherche au contraire à uniformiser les procédures d'expression des règles, des faits et des interfaces graphiques dans des tableurs. De plus, la fonctionnalité de règles externes n'est disponible que dans la version payante du produit. Comme les conditions et les actions des règles ne peuvent être exprimées qu'en Java, la note donnée sur ce critère est 0 sur 2.

5.2.2 Expressivité du langage de définition des règles

Drools

La possibilité de définir un DSL améliore grandement l'expressivité du langage. S'il est bien conçu, le DSL permet d'exprimer des notions du domaine d'affaires de manière claire et succincte, libre des contraintes de format des langages de programmation classiques. Prenons comme exemple l'entrée suivante dans un fichier .dsl :

```
[condition][] Il existe un [Cc]ompte qui = $compte : Compte()
[condition][] -possede un proprietaire = prop != null
[condition][] -possede un solde superieur a {montant} = solde > {montant}
[condition][] -utilise la monnaie {currency} = monnaie == {currency}
```

La partie gauche de l'équivalence définit une entrée dans le langage DSL. La partie droite définit son implémentation en code DRL, qui est la syntaxe par défaut d'expression de règles de Drools. [Cc] dans [Cc]ompte (première entrée) est une expression régulière qui signifie que les phrases "Il existe un compte qui" et "Il existe un Compte qui" sont toutes deux traduites en "$compte : Compte()". Les tirets au début de chacune des conditions indiquent que ces phrases peuvent suivre la première condition dans n'importe quel ordre, ce qui améliore l'expressivité et la flexibilité du langage. On pourrait par exemple utiliser le fichier .dsl précédemment défini dans un fichier .dslr de définition de règles comme suit :

```
Il existe un compte qui -utilise la monnaie CAD -possede un solde
    superieur a 400
```

Par ailleurs, Java et sa panoplie de bibliothèques permettent d'exprimer les règles les plus sophistiquées. Pour ceux que la verbosité du langage Java rebute, Drools permet de profiter de ses bibliothèques tout en utilisant Scala ou Groovy pour un code plus concis. L'accès à ces langages est toutefois en cours de développement. La variété des approches disponibles devrait permettre une bonne expressivité, en fonction des règles à exprimer et du profil des utilisateurs. Drools obtient la note de 2 pour ce critère.

CLIPS

Bien qu'il soit écrit en C, CLIPS propose son propre langage, non soumis aux contraintes syntaxiques du C. Le langage adopte une notation parenthésée (à la LISP) où les opérandes précèdent les opérateurs. Il fournit un ensemble de constructions et de commandes relatives à la production de règles (defrule, deffact..) et à l'approche objet (defclass, defmethod, send...), ainsi que des méthodes utilitaires (fonctions mathématiques, manipulation d'entiers et de chaînes de caractères...). À partir de ce modèle commun générique, on est capable de construire un vocabulaire spécifique au domaine du système.

Le code suivant définit un modèle de fait *Compte* et une règle *client-vip* avec le langage CLIPS :

```
;on definit le modele de fait pour un compte
(deftemplate Compte
        (slot proprietaire)
        (slot solde)
        (slot monnaie)
)

;on definit la regle pour afficher les clients VIP
( defrule client-vip
        ( Compte (proprietaire ?prop) (solde ?solde) (monnaie CAD) )
        ( test (> ?solde 200) )
=>
        (printout t "Le client " ?prop " est VIP" crlf)
)
```

Il est clair que la notation parenthésée et l'ordre des opérateurs font que le langage est moins proche du langage naturel que ce qu'on a obtenu précédemment avec le DSL Drools. Cependant, le fait que CLIPS soit spécialisé dans la construction de systèmes experts permet de se libérer des contraintes propres aux langages de programmation généraux tel que C et Java. Le langage ciblé permet aussi de directement insérer/rétracter des faits, afficher l'ensemble de règles de la base de connaissances ou encore le contenu de l'agenda du moteur de règles, en utilisant des commandes prédéfinies. De plus, le modèle déclaratif produit un code clair et concis, qui représente la base de connaissances de manière succincte. Le langage de CLIPS est donc expressif mais impose une structure rigide nécessitant un effort d'adaptation des règles telles qu'elles sont exprimées par les experts métier. Le produit obtient une note de 1 sur 2 pour ce critère.

OpenRules

Les règles sont exprimées dans OpenRules avec des tables décisionnelles. Cette approche peut être adéquate si les données de l'organisme sont déjà exprimées sous ce format. Toutefois, le format n'est adéquat que si plusieurs règles partagent les mêmes conditions. Les tables décisionnelles peuvent devenir trop nombreuses pour des ensembles de règles

utilisant des conditions différentes. Le format est également verbeux car chaque ligne de la table représente une règle alors qu'une seule règle, exprimée dans une autre approche, pourrait synthétiser plusieurs lignes.

Si on exprimait l'exemple de règle utilisé pour l'évaluation de Drools et CLIPS, on obtiendrait le résultat suivant :

	A	B	C	D
35				
36		Datatype Compte		
37		String	proprietaire	
38		Double	solde	
39		String	monnaie	
40				
41		Rules void clientVIP(Compte compte)		
42		C1	C2	A1
43		compte.solde > s	compte.monnaie.equals(m)	out.print("Le client " + c.proprietaire + " est" + statut);
44		Double s	String m	String statut
45		Si le solde est superieur a	Si la monnaie du compte est	Le client est
46		100	CAD	VIP
47				

Figure 5.3 Règle exprimée avec OpenRules

On remarque d'une part que les conditions et les actions sont exprimées dans un format impératif. Les tables décisionnelles ne sont qu'un maquillage graphique, mais le code reste essentiellement du code impératif Java. D'autre part, l'approche pourrait être adéquate si on avait plusieurs règles avec les mêmes conditions mais des valeurs différentes. Sinon, la table décisionnelle aura été créée pour contenir une seule ligne.
Ainsi, même si elles constituent un format populaire pour les utilisateurs métier, les tables décisionnelles présentent une expressivité limitée des règles d'affaires. De plus, OpenRules n'effectue qu'un arrangement "cosmétique" du code Java en table décisionnelle, tout en gardant sa nature impérative.
En revanche, OpenRules fournit des classes utilitaires qui permettent de définir les données avec une expressivité améliorée :

- Intervalles FromToInt : L'expression Java "(x > 100) (x <= 550)" peut être exprimée par : "more than 100 and less or equal to 550", "(100 ;550]" ou encore "greater than 100 but less 551"

- Valeurs CompareToInt, CompareToDouble... : Comparer des nombres de types différents. Par exemple, la valeur d'un attribut de type float peut directement être comparée avec un entier.
- Domaines DomainString, DomainInt, DomainDouble... : Saisir un ensemble de valeurs à l'intérieur d'une même cellule du tableur, au lieu de décomposer la cellule en autant de sous-cellules que d'éléments du domaine. Les éléments d'un domaine de chaînes de caractères sont séparés par un espace (A B C), ceux de nombres sont séparés par des virgules (24,36,72).
- Expressions Java : Saisir directement des expressions Java dans des cellules de la table décisionnelle. Les expressions sont placées entre accolades (c.soldeCarteCredit <= 0 ;) ou précédées de " :=" (:=c.soldeCarteCredit <= 0 ;)
- Les types prédéfinis FromToInt, CompareToInt, etc sont implémentés dans le paquetage *com.openrules.types*. Il est possible de personnaliser et d'étendre ce paquetage pour améliorer l'expressivité selon les besoins de l'organisme.

Les tables décisionnelles sont un outil populaire d'expression des règles. OpenRules offre des classes utilitaires améliorant l'expressivité de ces tables. Cependant, l'approche présente des limites et n'est pertinente que pour un type de règles bien particulier. Le produit obtient une note de 1 sur 2.

5.2.3 Prise en charge de l'internationalisation

Drools

Le langage Java offre une prise en charge avancée de l'internationalisation grâce aux bibliothèques i18n.

Si on ne désire pas utiliser du code Java pour exprimer les règles, il est toujours possible d'utiliser les modèles (templates) Drools et de définir des règles de transformation de données selon l'environnement. On pourrait entre autres réaliser une conversion automatique de sommes d'argent d'une monnaie vers une autre. Par exemple, la méthode *getConversionToEurFrom()* invoquée dans la règle suivante pourrait récupérer le taux

de change actuel à partir d'un service web.

```
rule conversionMonnaieVersEURO
        when
                $mapComptes : Map (this["_type_"] == "Compte",
                        this["monnaie"] != null && != "EUR")
                $montantConverti : String() from getConversionToEurFrom($
                    mapComptes["monnaie"])
        then
                modify( $mapComptes ) {
                        put("solde", $montantConverti * $mapCompte.get("
                            solde"))),
                        put("monnaie", "EUR")
                }
end
```

Drools peut utiliser les bibliothèques i18n de Java mais ne fournit pas ses propres outils de prise en charge de l'internationalisation. Il obtient la note de 1 pour ce critère.

CLIPS

CLIPS ne prend en charge que des chaînes de caractères composées de symboles du jeu de caracères anglais standard (US7ASCII). Pour prendre en charge l'internationalisation, il faudrait passer par un langage de programmation s'interfaçant avec CLIPS. On pourrait alors échanger les données avec CLIPS exclusivement en anglais et réaliser les transformations avec le langage de programmation tiers. Ces transformations seraient réalisées par des méthodes à définir par une équipe de développement et pourrait s'appuyer sur les bibliothèques de prise en charge de l'internationalisation proposées par le langage tiers (Python ou Java par exemple). Ainsi, la gestion de l'internationalisation n'est pas prise en charge par CLIPS. Son implémentation demande un certain effort de développement et reste totalement découplée du système de production de règles. Le produit obtient 0 sur 2 pour ce critère, car il ne le satisfait pas.

OpenRules

OpenRules peut prendre en charge l'internationalisation sur plusieurs niveaux :

- Les règles étant définies en Java, on est capable d'utiliser les bibliothèques i18n du langage.
- Les types prédéfinis OpenRules du paquetage *com.openrules.types* (mentionnés dans la section précédente) peuvent être étendus. Étant écrites en Java, ces classes utilitaires pourraient être adaptées selon le langage local, grâce aux bibliothèques i18n.
- Définir des règles d'internationalisation dans des tableurs. Le format s'y prête bien : chaque ligne pourrait représenter un langage et chaque colonne un attribut "internationalisé".

Ainsi, OpenRules permet de prendre en charge l'internationalisation, mais ne fournit pas d'outils prédéfinis. Il obtient une note de 1.

5.2.4 Compréhension du modèle de données

Drools

Drools permet d'importer des classes et des paquetages de classes représentant un modèle de données défini en Java. L'import se fait avec des instructions *import*, similaires à celles de Java, dans les fichiers de définition des règles.

```
import banque.modele.Addresse;
import banque.modele.Compte;
```

Drools permet aussi de définir des entités directement dans les définitions des règles et de les insérer dans la base de connaissances pour être traitées.

```
declare Personne
 nom : String
 prenom : String
 age : int
end
```

Le premier choix offre des possibilités considérables de réutilisation des modèles de données existants. En effet, les classes Java, via des bibliothèques tel que iBatis ou Hibernate, peuvent former des passerelles vers une panoplie de sources de données (bases de données relationnelles, fichiers XML, fichier plats, dépôts LDAP...).
Drools obtient la note de 2 pour ce critère.

CLIPS

CLIPS permet de définir un modèle de données avec des modèles de faits, similaires aux structures C, via la construction *deftemplate*.

```
; Definition de la structure compte
 (deftemplate compte
        (slot numero)
        (slot proprietaire)
        (slot solde) ; prend 0 ou une valeur
        (multislot transactions) ; prend 0..n valeurs
)

; Insertion d'un fait de type compte
(assert (compte
        (numero 12324323)
        (proprietaire jihed)
        (solde 232334)
        (transactions 2342 -6532 4235)))
```

Pour l'approche objet, COOL (CLIPS Object Oriented Language) permet de définir des classes et des instances de classes via les constructions *defclass* et *definstance*. OBJECT est la classe racine du système de classes CLIPS et USER est une classe prédéfinie, sous-classe de OBJECT.

```
(defclass COMPTE (is-a USER)
        (slot numero (default (gensym*))) ; gensym* genere un numero
            unique pour chaque nouvelle instance de la classe
        (slot proprietaire (create-accessor read-write)) ;l'accesseur et
            le mutateur de l'attribut sont generes par CLIPS
```

```
        (slot solde)
        (multislot transactions)
)

; Creation d'une instance de la classe COMPTE
(make-instance comptel of COMPTE (proprietaire jihed) (solde 2343523))
```

Le modèle du langage CLIPS présente un couplage faible entre les règles et les objets. Les règles peuvent communiquer avec les objets via le passage de messages, mais elles ne peuvent pas être facilement imbriquées dans des objets.

D'autre part, en interfaçant CLIPS avec un langage tiers (C/C++, Ada, Python...), on est capable d'affecter aux structures et aux instances de classes CLIPS des valeurs provenant d'une source externe tel qu'une base de données relationnelle, un fichier plat ou n'importe quelle autre source de données accessible à partir de ce langage. Si l'interfaçage avec Java fournit par la bibliothèque CLIPSJNI reste très primitif, celui fournit par PyCLIPS pour le langage Python est assez mature. Un exemple exhaustif publié sur le site du projet illustre l'accès à une base de données relationnelle [11]. L'exemple définit une stratégie de vente de voitures. Les règles et les modèles de faits sont définis avec CLIPS. Des tables, créés dans un système de gestion de bases de données relationnelle, stockent des enregistrements de voitures, de villes et de contrats. À l'aide d'une méthode Python, on récupère les enregistrements de la BD avec une requête SQL et on affecte les valeurs aux faits CLIPS.

Ainsi CLIPS permet à la fois de définir un modèle de données interne au système et d'accéder indirectement à des sources de données externes. L'effort pour réaliser cette deuxième fonctionnalité est double : un premier lien doit être établi entre la source de données et le langage tiers ; les résultats sont ensuite transformés en faits et en instances de classes CLIPS. Contrairement à Drools qui permet de manipuler des entités Java à partir des règles, CLIPS ne peut pas directement accéder aux entités définies dans un

11. http ://pyCLIPS.sourceforge.net/web/ ?q=node/25 (dernière consultation le 09-11-10)

langage tiers (C/C++, Python ou autre). Il obtient une note de 1 pour ce critère.

OpenRules

OpenRules permet de définir des faits "à la volée" dans des tableurs ou encore d'accéder à des entités définies dans des fichiers XML ou dans des classes Java :

- Tableurs : OpenRules s'inspire des bases de données relationnelles pour définir les données dans des tableurs.

  Un nouveau type de données est déclaré dans une table *DataType*. Chaque ligne d'une table DataType correspond à un attribut du nouveau type. Une ligne est composée de deux colonnes : la première définit le type de l'attribut et la seconde définit son nom.

Datatype Client	
String	nom
String	sexe
String	etatCivil
int	age

Data Client clients			
nom	sexe	etatCivil	age
Nom	Sexe	Etat Civil	Age
Zeineb	Feminin	Marie(e)	55
Jihed	Masculin	Celibataire	27

Figure 5.4 Données définies dans un tableur

  Les données sont définies dans une table *Data*. Dans l'exemple de la figure 5.4, *Data Client clients* déclare une table de données nommée clients et composée d'éléments de type *Client*.

  Le premier attribut constitue la clé primaire d'un enregistrement.

  On peut donc récupérer le client de nom "Zeineb" comme suit :

```
Client c1 = clients["Zeineb"];
```

- Fichiers XML : Il est possible de définir des données dans des fichiers XML. Le fichier XML peut contenir plusieurs entrées :

```
<Clientele type="Array of Client(s)">
        <Client nom="Zeineb" sexe="Feminin" etatCivil="Marie(e)" age=
            "55" />
        <Client nom="Jihed" sexe="Masculin" etatCivil="Celibataire"
            age="27" />
```

```
</Clientele>
```

Le type *Array of* est un type prédéfini OpenRules. *Array of Client(s)* indique à OpenRules que la balise *Clientele* contient plusieurs balises *Client*. À l'extraction des données, OpenRules génère une classe Java nommée *Client* (dont les attributs sont déduits de la balise *Client*) et une classe *Clientele*, tableau d'éléments de type *Client*. L'accès aux données se fait comme suit :

```
Clientele clientele = Clientele.load("classpath:/FichierClients.xml");
for (int i = 0; i < clientele.Client.length; i++) {
    Clientele.Client client = clientele.Client[i]
}
```

FichierClients.xml est le nom du fichier XML où les clients sont définis et *load* est la méthode utilitaire OpenRules qui réalise l'extraction des données à partir du fichier XML.

Environment	
import.static	com.openrules.tools.Methods
import.schema	classpath:/Client.xml
import.java	banque.*
include	../include/Regles.xls

Figure 5.5 Import de données Java, XML et XLS avec OpenRules

- Classes Java : OpenRules permet d'importer des classes et des paquetages de classes représentant un modèle de données défini en Java. L'import se fait avec des instructions *import*, similaires à celles de Java, dans des tables de type *Environment*. *import.static* permet d'importer des membres statiques et de les utiliser directement dans les tableurs. Les entités définies dans des classes Java peuvent constituer, à leur tour, des passerelles vers une panoplie de sources de données externes. Par exemple, un tableur OpenRules importerait une classe Java nommée *Comptes* qui récupère les données des instances de comptes à partir d'un répertoire LDAP.

Ainsi, OpenRules peut manipuler des entités du modèle de données définies dans plusieurs formats. Les bibliothèques d'extraction et de transformation des données disponibles sont nombreuses et faciles d'accès. Le produit obtient une note de 2 sur 2.

5.3 Fonctions facilitant l'expression des règles

5.3.1 Modèles de règles

Drools

Drools offre des modèles de règles basés sur une approche recherche/remplacement de valeurs. Pour illustrer le fonctionnement des modèles de règles Drools, on définit un modèle dans un fichier qu'on nomme *template.dlr*

```
template header
type
solde
limite

package compte;

template "compte"
rule "est de type @{type}"
        when
                Item( $type : @{type} , $solde : @{solde} <= @{limite} )
        then
                System.out.println("Compte " + $type + "@{type} avec solde
                    " + $solde + " inferieur a @{limite}");
end

end template
```

L'en-tête (template header) liste les champs variables du modèle. Les modèles de règles sont ensuite définis, les champs variables y sont utilisés selon le format @nom du paramètre.

Pour tester ce modèle, on crée une méthode de test Java qui passe au modèle deux ensembles de valeurs pour ses champs variables.

```
Collection<ParamSet> paramSets = new ArrayList<ParamSet>();
// populate paramSets
```

```
paramSets.add(new ParamSet("Etudiant", 424.32, 10000));
paramSets.add(new ParamSet("Homme d'affaires", 12334.33, 400000));
ObjectDataCompiler converter = new ObjectDataCompiler();
InputStream templateStream = this.getClass().getResourceAsStream("
    template.drl");
String drl = converter.compile(paramSets, templateStream);
System.out.println(drl);
```

L'exécution de la méthode de test produit deux instances de règles basées sur le même modèle. Les valeurs passées en paramètre sont affectées aux champs variables.

```
rule "est de type Homme d'affaires"
        when
                Item( $type : Homme d'affaires , $solde : 12334.33 <=
                    400000.0 )
        then
                System.out.println("Compte " + $type + "Homme d'affaires
                    avec solde " + $solde + " inferieur a 400000.0");
end

rule "est de type Etudiant"
        when
                Item( $type : Etudiant , $solde : 424.32 <= 10000.0 )
        then
                System.out.println("Compte " + $type + "Etudiant avec
                    solde " + $solde + " inferieur a 10000.0");
end
```

Ainsi, les modèles de règles sont implémentés dans Drools, mais leur fonctionnalité se limite à une approche recherche/remplacement basique. La note obtenue pour ce critère est donc 1.

CLIPS

Le langage CLIPS ne permet pas de générer une règle en passant des valeurs aux paramètres variables d'un modèle de règles. Avec CLIPS, une règle est toujours définie

à partir de zéro. On pourrait simuler la fonctionnalité avec un langage tiers, mais le concept de modèle de règle, l'instanciation de règles à partir de modèles et le passage du résultat à CLIPS devraient être développés en dehors du système. Ainsi, CLIPS ne satisfait pas le critère et obtient la note zéro.

OpenRules

À partir de la version payante 5.0 d'OpenRules, il est possible de définir l'en-tête des tables décisionnelles dans un modèle. L'en-tête définit la signature de la règle (nom et types de faits passés en paramètre) ainsi que ses conditions et ses actions. À partir de ces modèles, on définit des tables où différentes valeurs sont affectées aux variables du modèle. Dans la figure 5.6, *Rules statutClientProvinceA implements statutClient* signifie que la table décisionnelle "statutClientProvinceA" utilise le modèle "statutClient".

Rules void statutClient(Compte compte)		
C1	C2	A1
compte.solde > s	compte.monnaie.equals(m)	out.print("Le client " + c.proprietaire + " est" + statut);
Double s	String m	String statut
Si le solde est superieur a	**Si la monnaie du compte est**	**Le client est**

Rules statutClientProvinceA **template** statutClient		
250	CAD	Regulier
600	CAD	Important
900	CAD	VIP

Figure 5.6 Modèle de règles OpenRules

Il est possible d'affecter aux variables du modèle des valeurs par défaut. Par ailleurs, les tables décisionnelles peuvent sélectionner les conditions et actions du modèle qu'elles implémentent :

- en les désignant par leurs identifiants du modèle.
- en spécifiant dans le modèle même que ces colonnes sont optionnelles ; les tables désirant les utiliser devront les désigner explicitement.

Les modèles OpenRules ne peuvent être réutilisés pour créer d'autres modèles. De plus, ils n'existent que pour la version payante d'OpenRules qui, bien que libre (à code ouvert

et utilisant une licence libre), rajoute un coût financier pouvant rebuter les organismes partenaires de la Chaire. Ce bilan mitigé vaut une note de 1 sur 2.

5.3.2 Procédures et algorithmes

Drools

Il est possible de définir avec Drools des fonctions à l'intérieur des fichiers de définition des règles comme suit :

```
function int anneesEcouleesDepuis(Date date) {
        return Years.yearsBetween(new DateMidnight(date), new DateMidnight
            ()).getYears();
}
```

On peut également importer des méthodes définies dans des classes Java. Ces méthodes peuvent manipuler des variables globales de la base de connaissances via l'API de Drools.

Le code suivant déclare deux variables globales et importe une classe comprenant des fonctions statiques dans un fichier .drl :

```
global ValidationReport validationReport;
global ReportFactory reportFactory;

import function validation.service.ValidationHelper.*;
```

Les fonctions, déclarées dans les fichiers .drl ou importées de classes Java, peuvent être invoquées dans les conditions et les conséquences des règles.

Il est également possible d'utiliser la fonction *eval* pour exécuter des instructions simples retournant une valeur booléenne.

```
rule "ageNonValideCompteEtudiant"
    when
       $client : Client( eval ( anneesEcouleesDepuis(dateDeNaissance) >=
           27 ) )
```

```
        $compte : Compte( type == Compte.Type.ETUDIANT ) and Compte(
            proprietaire == $client )
    then
        error(kcontext, $compte);
end
```

Si on ajoute la possibilité d'utiliser le langage objet-fonctionnel Scala ou encore le langage CLIPS, on comprend que Drools mérite une note de 2 pour ses capacités de définition et d'appel de fonctions.

CLIPS

CLIPS offre plusieurs façons de déclarer des fonctions :

- Fonctions définies dans le langage natif CLIPS : *deffunction* définit le nom de la fonction, la liste de paramètres requis, un paramètre optionnel qui représente un ensemble variable de paramètres et la liste d'instructions à exécuter lorsque la fonction est appelée.

```
(deffunction afficher-parametres (?a ?b $?c)
   (printout t ?a " " ?b " et " (length ?c) " parametres optionnels: " ?
    c crlf))

; Appel de la fonction afficher-parametres
CLIPS>(afficher-parametres a b c d)
a b et 2 parametres optionnels: (c d)
```

  Le langage CLIPS prend en charge les appels récursifs, les boucles d'itérations et les structures de contrôle.

- Méthodes définies dans le langage natif CLIPS : *defgeneric* est similaire à *deffunction* mais offre en plus la surcharge des opérateurs. L'instruction *defgeneric* déclare l'en-tête de la fonction. Un ensemble d'implémentations de la méthode générique est créé avec *defmethod*. Une de ces méthodes est sélectionnée en fonction du type des paramètres passés à l'exécution.

```
; Declaration de l'en-tete avec defgeneric
```

```
(defgeneric >)

; Surcharge de la methode ">"
(defmethod > ((?a STRING) (?b STRING))
        (> (str-compare ?a ?b) 0))

CLIPS> (> "chaine2" "chaine1") ; appel de la methode ">" avec deux
    parametres STRING
TRUE ;l'appel retourne que "chaine2" > "chaine1"
```

- Messages définis dans le langage natif CLIPS : *defmessage-handler* spécifie le comportement d'une classe d'objets CLIPS en réponse à un message donné. L'implémentation du message est composée de code procédural.

```
; On declare une classe A
(defclass A (is-a USER))

; On declare la methode afficher-parametres de la classe A
(defmessage-handler A afficher-parametres (?a ?b $?c)
   (printout t (instance-name ?self) " " ?a " " ?b " et " (length$ ?c) "
        parametres optionnels: " ?c crlf))

CLIPS> (make-instance a of A) ; creation d'une instance de la classe A
[a]
CLIPS>(send [a] afficher-parametres a b c d) ; appel de la methode
    afficher-parametres
[a] a b et 2 parametres optionnels: (c d)
```

- Fonctions définies dans un langage externe tel que C ou Python : Il est possible de déclarer des méthodes dans un langage de programmation tiers (Python, c..) et de les appeler à partir de CLIPS. Dans l'exemple suivant, on déclare une fonction Python appelée *py_carre* qui retourne le carré d'un entier passé en paramètre. La méthode est ensuite appelée à partir de CLIPS avec la commande *python-call*. L'exemple utilise la bibliothèque PyCLIPS d'interfaçage du langage Python avec CLIPS.

```
import CLIPS as CLIPS
```

```
def py_carre(x):
    return x * x

if __name__ == '__main__':
    CLIPS.RegisterPythonFunction(py_carre)
    CLIPS.SetExternalTraceback(True)
    print CLIPS.Eval("(python-call py_carre 7)")
```

Ainsi, CLIPS prend en charge la déclaration de fonctions et l'appel de fonctions externes. Il obtient la note de 2 sur 2 pour ce critère.

OpenRules

OpenRules est étroitement lié au langage Java. Les conditions et les actions des règles, les types de données et le moulinage des tableurs sont exprimés en Java. De ce fait, OpenRules ne "permet" pas d'invoquer des fonctions mais il est carrément fondé sur l'utilisation d'instructions et de méthodes codées en Java. Dans une table de type *Environment*, *import:static* permet d'importer des fonctions statiques Java, comme celles définies dans les classes utilitaires du paquetage *com.openrules.tools.Methods. import.java*, suivie du nom des classes ou des paquetages Java à importer, permet de les utiliser directement dans les tableurs.
De plus, les tables de type *Method* permettent de définir des méthodes Java directement dans les tableurs, comme le montre la figure ci-dessous.

9	{ customer.monthlyIncome * 0.8 - customer.monthlyDebt > loan.amount/loan.term; }
10	
11	**Method String[] getCustomers()**

12	int n = customers.length;
	String[] names = new String[n];
	for (int i=0; i<n; i++)
	  names[i] = customers[i].fullName;
	return names;

13	

Figure 5.7 Déclaration d'une méthode dans OpenRules

OpenRules permet donc de définir et d'invoquer des fonctions à partir des tables décisionnelles et obtient la note de 2 sur 2 pour ce critère.

5.3.3 Flux de règles

Drools

Drools possède un module sophistiqué de gestion du flux de travail nommé *Drools Flow* ou *Ruleflow*. Les processus définis avec *Drools Flow* permettent les branchements conditionnels, la gestion des exceptions et la définition d'actions personnalisées. On illustre la gestion du flux de règles Drools avec une séquence de traitement de demande d'emprunt, inspirée de Bali (Bali, 2009).

La première étape consiste à vérifier que la demande de prêt est valide. Si elle ne l'est pas, un courriel est envoyé au client pour l'en informer et le flux se termine. Sinon, si le montant demandé est important, calculer le score du client. S'il est faible, en informer le client par courriel et achever le traitement de la demande. Sinon, attendre un événement d'approbation pour transférer les fonds, informer le client de l'opération et achever le traitement avec succès.

Il est à noter que :

- des variables de flux sont définies comme suit :

```
<variable name="customer" >
        <type name="org.drools.process.core.datatype.impl.type.
            ObjectDataType" className="droolsbook.bank.model.Customer" />
</variable>
```

- les nœuds d'évaluation obéissent à des règles d'affaires. Le bout de code suivant oriente le flux selon l'importance du montant de prêt demandé :

```
<nodes>
    <start id="1" name="Start" x="191" y="86" />
    <end id="3" name="Fin" x="345" y="667" />
    <split id="4" name="Montant a emprunter?" x="227" y="229" type="2" >
      <constraints>
        <constraint toNodeId="7" name="sinon" priority="100" type="rule"
            dialect="mvel" >eval(true)</constraint>
        <constraint toNodeId="5" name="petit" priority="1" type="rule"
            dialect="mvel" >Pret( montant &lt;= 5000 )</constraint>
```

```
    </constraints>
  </split>
```

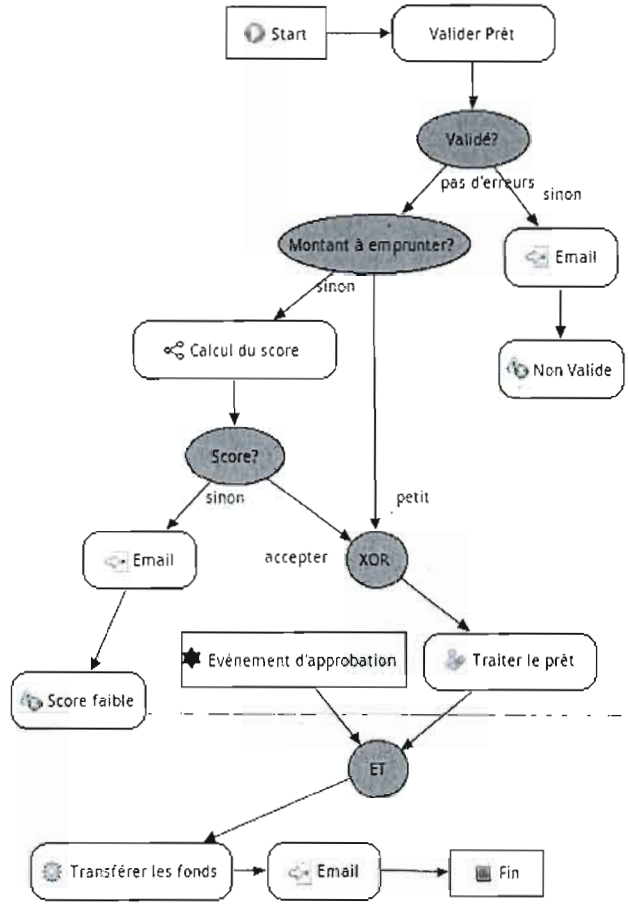


Figure 5.8 Flux Drools Flow de traitement d'une demande d'emprunt (Bali, 2009)

- L'utilisateur peut définir ses propres tâches personnalisées comme l'élément *Email* de l'exemple suivant. La tâche *Email* prend un certain nombre de paramètres et génère un courriel qu'elle envoie à un destinataire spécifié.

```
<workItem id="25" name="Email" x="385" y="238">
  <work name="Email" >
    <parameter name="Corps" >
      <type name="org.drools.process.core:datatype.impl.type.
          StringDataType" />
```

```
  <value>Cher #{client.prenom},
Il manque les donnees suivantes dans la demande de pret que vous
    avez envoye:
#{errorList}
Veuillez reverifier votre demande et nous la reacheminer.
```

- le calcul du score est un sous-processus Drools Flow qui retourne un résultat de succès ou d'échec au processus principal de traitement de demandes d'emprunt.
- Les règles appartiennent à des groupes de règles. Par exemple, la validation d'une demande de prêt invoque un ensemble de règles de validation qui appartiennent au groupe de règles nommé *validerPret*.

```
<ruleSet id="14" name="Valider Pret" x="115" y="40" ruleFlowGroup="
    validerPret" />
```

La règle *montantNonValide* fait partie du groupe de règles à déclencher à l'étape de validation de la demande de prêt.

```
ruleflow
rule montantNonValide
ruleflow-group "validerPret"
        when
                Pret(montant == null || amount < 0)
        then
                insert(new DefaultMessage());
end
```

Drools offre une prise en charge avancée des flux de règles. Il obtient une note de 2 pour ce critère.

CLIPS

CLIPS permet de contrôler le flux d'exécution des règles en utilisant les modules, l'insertion et la rétraction de faits ou, à moindre degré, la priorité des règles.

- *defmodule* : permet de structurer la base de connaissances en modules. Chaque module contient un ensemble de constructions et définit ses propres restrictions d'accès. Il possède son propre agenda et son propre réseau de modèles d'appariement pour

les règles qu'il contient. Une commande *run* exécute l'agenda du module actuellement sélectionné. Les commandes *reset* et *clear* sélectionnent le module principal MAIN. Lorsqu'un nouveau module est sélectionné, l'exécution des règles du module précédemment sélectionné s'arrête. Lorsque l'agenda du module sélectionné ne contient plus de règles, le prochain module dans la pile des modules est sélectionné. Avant l'exécution d'une règle, le module dans lequel elle a été définie est sélectionné. Le module sélectionné peut être modifié avec la commande *focus*. Le code suivant illustre le contrôle de flux par modules.

```
; Le module principal exporte ses constructions aux autres modules
(defmodule MAIN(export ?ALL))
; On definit la regle r1 du module principal qui selectionne les modules
    A puis B.
(defrule MAIN::r1
=>
(printout t "Execution des regles du module Principal." crlf)
(focus A B))
; On definit le module A qui importe les faits du module principal
(defmodule A (import MAIN deftemplate initial-fact))
; On definit la regle r2 du module A
(defrule A::r2
=>
(printout t "Execution des regles du module A." crlf))
; On definit le module B qui importe les faits du module principal
(defmodule B (import MAIN deftemplate initial-fact))
; On definit la regle r3 du module B
(defrule B::r3
=>
(printout t "Execution des regles du module B." crlf))
; On selectionne le module principal et on execute les regles
CLIPS> (reset)
CLIPS> (run)
Execution des regles du module Principal.
Execution des regles du module A.
Execution des regles du module B.
```

- Faits de contrôle : Une opération complexe est divisée en un flux de sous-tâches. Une règle n'est activée que lorsqu'un fait de contrôle donné existe dans la base de faits. À la fin de son exécution, la règle retire de la base de faits le fait responsable de son activation et insère un nouveau fait de contrôle qui sert de condition à l'activation de la tâche suivante dans le flux. Le principe d'insertion/rétraction de faits de contrôle continue jusqu'à atteindre la dernière règle du flux. L'exemple suivant permet de visualiser le principe :

```
(defrule r1
        ?fc1 <- (fait-controle1)
=>
        (printout t "Execution de r1" crlf)
        (retract ?fc1)
        (assert (fait-controle2))
)

(defrule r2
        ?fc2 <- (fait-controle2)
=>
        (printout t "Execution de r2" crlf)
        (retract ?fc2)
)

CLIPS>(reset)
CLIPS>(assert (fait-controle1))
CLIPS>(run)
Execution de r1
Execution de r2
```

- Priorité des règles : L'attribut *salience* d'une règle permet de lui affecter un niveau de priorité. Lorsque plusieurs règles existent dans l'agenda, celle avec la plus haute priorité est exécutée en premier. La valeur de l'attribut *salience* doit être entre -10000 (priorité la plus basse) et 10000 (priorité la plus élevée). La valeur par défaut est 0.
- (Chinn et Madey, 1999) permet d'ajouter une dimension temporelle aux faits. L'extension fournit une fonction *timerel* prenant en paramètre un fait, une date de début

et une date de fin du fait. L'article illustre l'utilité de l'extension en implémentant un flux de travail où une équipe d'ingénieurs réalisent un ensemble de tâches selon un flux donné.

Ainsi, plusieurs techniques permettent d'exprimer des flux de règles avec CLIPS, aussi bien à l'aide de constructions du langage que d'extensions. Le produit obtient 2 sur 2 pour ce critère.

OpenRules

OpenRules présente les flux de traitement avec des tables de transition d'état. Les figures 5.9, 5.10 et 5.11 résument l'exemple de validation de revenu pour une demande d'emprunt [12] de la documentation d'OpenRules, illustrant le mécanisme de flux.

- La figure 5.9 montre une table de type *Layout*, à partir de laquelle OpenRules génère une page HTML avec du code JSP.
- La table de la figure 5.10 établit les règles de navigation dans l'application web. Chaque étape de l'interaction utilisateur-application possède une étape précédente et une étape suivante.
- La figure 5.11 définit les règles à exécuter et les fenêtres à afficher à chaque étape de l'interaction.

Layout TableLayout FenetreChercherClient(DemandeEmprunt emprunt)	
"L'emprunteur est il un client connu?"	[emprunt.clientConnu]["true,Oui;false,Non"]
"Sélectionnez l'emprunteur dans la liste de clients:"	[emprunt.client.nomComplet][getClients()]

Figure 5.9 Définition d'une fenêtre OpenRules

Le code utilisé pour les conditions et les actions des deux tables de gestion du flux aux figures 5.10 et 5.11 est un code impératif Java avec des structures de contrôle classiques. En effet, il ne s'agit pas de définir des groupes de règles éligibles à être exécutées à une

12. http ://openrules.com/docs/man_forms.html#How OpenRules Validators Work (dernière consultation le 09-11-10)

Rules void **reglesFluxTraitement(DemandeEmprunt emprunt)**				
C1	C2	A1	A2	A3
dialog().isCurrent(step)	condition	if (dialog().isInitial()) dialog().next = etape;	if (dialog().isAction("Suivant")) dialog().next = etape;	if (dialog().isAction("Precedent")) dialog().next = etape;
String step	boolean condition	String etape	String etape	String etape
Action **Etapes traitement et conditions**		**Init**	**Suivant**	**Precedent**
		DonneesGenerales		
DonneesGenerales			DonneesEmprunt	DonneesGenerales
DonneesEmprunt			ChercherClient	DonneesGenerales

Figure 5.10 Définition de l'étape précédente et suivante de chaque fenêtre

Rules void **nextLayout(DemandeEmprunt emprunt, Client client)**	
C10	A10
dialog().isNext(step)	dialog().nextLayout = layout
String step	TableLayout layout
Si la prochaine etape est	**Alors executer les regles et definir l'etape suivante**
DonneesGenerales	{ FenetreDonneesGenerales(client); }
DonnesEmprunt	{ FenetreDonneesEmprunt(emprunt); }
ValiderResultatsRevenu	{ reglesValidationRevenu(emprunt, client); FenetreResultatsValidationRevenu(emprunt); }

Figure 5.11 Règles et fenêtre à déclencher à chaque étape

étape donnée comme le font Drools et CLIPS mais d'exécuter une succession de blocs *if..else*. Cette approche est limitée et n'apporte pas de valeur ajoutée. Elle ne fournit pas la possibilité de faire des convergences et des divergences parallèles et synchronisées de gestion de flux comme c'est le cas avec Drools Flow.

Ainsi OpenRules permet de gérer les flux de règles, mais sans apporter de valeur ajoutée au code impératif Java. Il obtient la note de 1 sur 2.

5.3.4 Tables et arbres de décision

Drools

Drools accepte des règles définies dans des tables décisionnelles, qu'elles soient sous format OpenOffice, Excel ou même de simples feuilles CVS (comma-seperated values).

De plus, l'interface d'administration web Guvnor de Drools permet de définir des tables décisionnelles à partir du navigateur web.

On illustre les capacités de Drools avec la définition des règles de calcul du taux d'intérêt suivante, inspirée du livre "Drools JBoss Rules 5.0 Developer's Guide" (Bali, 2009).

	A	B	C	D	E
1	**RuleSet**	tablesdecisionelles			
2	**Import**	banque.modele.Client. banque.modele.Compte.Type. java.math.*.			
3	**Notes**	Tables decisionelles pour le calcul du taux d'intérêt			
4					
5	Table de règles de calcul d'intérêt				
6	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
7	$a:Compte				
8	type == Compte.Type. $param	monnaie	solde >= $1 && < $2	moisEntreDateDebutEtFin >= $1 && < $2	$a.setTauxInteret(new BigDecimal($param));
9	**type**	**monnaie**	**solde (min,max)**	**mois**	**taux d'intérêt**
10	TRANSACTIONNEL	EUR			"0.01"
11	ETUDIANT	EUR	0, 2000		"1.00"
12	EPARGNE	EUR	0, 100	0, 1	"0.00"
13	EPARGNE	EUR	0, 100	1, 3	"0.10"
14	EPARGNE	EUR	0, 100	3, 12	"2.00"
15	EPARGNE	EUR	100, 1000	0, 1	"0.10"
16	EPARGNE	EUR	100, 1000	1, 3	"3.00"
17	EPARGNE	EUR	100, 1000	3, 12	"3.25"
18	EPARGNE	EUR	1000, 5000	0, 1	"0.10"
19	EPARGNE	EUR	1000, 5000	1, 3	"3.25"

Figure 5.12 Table décisionnelle de règles de taux d'intérêt (Bali, 2009)

Les colonnes CONDITION sont fusionnées car l'objet *Compte* est utilisé dans toutes les conditions. Les paramètres $1 et $2 pour les colonnes *solde* et *mois* représentent les deux valeurs séparées par une virgule. La règle de la colonne *monnaie* est automatiquement traduite en "monnaie = la valeur en paramètre". La première ligne de notre table signifie que pour chaque compte transactionnel avec une monnaie EUR (euro), affecter un taux d'intérêt de 0.01% (quelque soit le solde du compte). La ligne 16 signifie : Pour chaque compte d'épargne dont le solde est entre 100 et 1000 euros et qui a été ouvert pour un à trois mois, affecter un taux d'intérêt de 3%. La règle suivante est générée pour la ligne 16 :

```
rule "Taux Interet_16"
```

```
when
$a:Compte(type == Compte.Type.EPARGNE,
monnaie == "EUR", solde >= 100 && < 1000,
moisEntreDateDebutEtFin >= 1 && < 3)
then
$a.setTauxInteret(new BigDecimal("3.00"));
end
```

Sans tables décisionnelles, on aurait dû écrire les règles manuellement. Drools permet de définir des tables décisionnelles dans plusieurs formats. On est capable de définir des règles complexes via ces tables Drools. Le produit obtient donc une note de 2 pour ce critère.

CLIPS

CLIPS ne prend pas directement en charge les tables décisionnelles. La fonctionnalité peut être implémentée de plusieurs manières :

- écrire une macro dans un tableur qui extrait les données des cellules et les écrit dans un fichier texte selon la syntaxe CLIPS. À chaque fois que le contenu de la table décisionnelle change, la macro doit être ré-exécutée pour recréer le fichier CLIPS.
- sous l'environnement Windows, écrire un composant COM et l'utiliser à partir d'une fonction définie dans CLIPS. Le composant utiliserait l'interface d'automatisation d'Excel pour lire la table décisionnelle et accéder aux cellules. La programmation du composant est non triviale mais permet d'accéder directement à la table décisionnelle.
- passer par un des langages qui s'interfacent avec CLIPS et utiliser les bibliothèques de manipulation de tableurs offertes par le langage.

Dans tous les cas, la transformation des données doit être entièrement réalisée par un développeur et demande un effort de codage considérable. À notre connaissance, il n'existe pas d'extension prédéfinie publiquement disponible qui fournisse cette fonctionnalité. Le produit obtient donc une note de 0 sur 2.

OpenRules

OpenRules permet de définir dans des tables décisionnelles non seulement les règles, mais aussi les types de données et les données ainsi que les interfaces web d'interaction avec l'utilisateur. Les formats de tableurs Microsoft Excel, OpenOffice et Google Spreadsheets sont officiellement pris en charge par le produit, qui se base sur la bibliothèque Java Apache POI pour accéder aux tableurs. Le produit satisfait donc ce critère et obtient une note de 2 sur 2.

5.4 Accessibilité aux experts métier

5.4.1 Accessibilité du langage aux non informaticiens

Drools

Tout en permettant l'écriture de code Java, Drools offre une couche d'abstraction du langage DRL et des dialectes Java et MVEL via la définition de DSLs. Il est possible, avec un DSL bien conçu, d'exprimer les règles dans un langage proche du langage naturel. L'invocation d'une séquence d'instructions informatiques est alors encapsulée et seule la notion métier correspondante est exposée à l'utilisateur. Ainsi, les utilisateurs métier n'ont pas besoin de formation à la syntaxe d'un langage de règles ou d'un langage de programmation bas niveau. L'utilisation d'expressions régulières pour faire correspondre un ensemble de phrases du DSL à leur implémentation en code assouplit encore plus les contraintes de syntaxe du langage.

À titre d'illustration, nous définissons les équivalences ci-dessous dans le DSL :

```
[keyword][] [Ll]es demandes de pret (du client|de) {client} sont non
    valables =
[Ll]e client {client} n'est pas eligible pour une demande d'emprunt
[condition][] [Ll]e client {client} n'est pas eligible pour une demande d'
    emprunt =
$client : Client( nom = {client})
$compte : Compte( proprietaire = $client && solde <= 400 )
```

l'ensemble des expressions suivantes est accepté dans les fichiers .dslr :

```
Les demandes de pret de Michael Jackson sont non valables
les demandes de pret du client Michael Jackson sont non valables
le client Michael Jackson n'est pas eligible pour une demande d'emprunt
```

et elles sont traduites en code .drl exécutable par Drools :

```
$client : Client( nom = "Michael Jackson")
$compte : Compte (proprietaire = $client && solde <= 400)
```

Drools obtient 2 pour ce critère car il permet de définir un langage d'expression de règles accessible aux non informaticiens.

CLIPS

Comme constaté précédemment, le langage CLIPS permet d'obtenir un code succinct et clair grâce à l'utilisation d'un style déclaratif et d'un ensemble de constructions prédéfinies ciblées pour la construction de systèmes experts. Les faits sont créés avec des instructions simples tel que

```
( assert(compte est-a-decouvert-pour client1))
```

Cependant, la notation à la LISP n'est pas la plus intuitive pour un utilisateur métier. Certaines constructions nécessitent également des compétences en développement (tel que la commande *bind* utilisée pour la manipulation de variables). Globalement, le langage CLIPS est accessible aux utilisateurs métier. Il reste toutefois loin du langage naturel et requiert du temps d'apprentissage. Ce bilan mitigé vaut une note de 1 sur 2.

OpenRules

OpenRules a été pensé dès sa conception comme un outil de gestion de règles accessible aux utilisateurs métier. La documentation du produit montre un scénario de définition d'un système de diagnostic médical où un technicien informatique et un médecin se partagent la création de l'application. Le médecin saisit les règles et des esquisses d'in-

terfaces graphiques dans un tableur et le technicien rajoute le code Java correspondant. Les tableurs ont d'ailleurs été choisis comme support de définition de tous les aspects du système car il s'agit d'un format populaire et accessible aux non informaticiens. Des classes utilitaires fournies par OpenRules pour définir des listes, des intervalles ou encore comparer des nombres de types différents rendent la saisie de données plus accessible aux utilisateurs métier. Ceci étant dit, la logique qui régit les données des tableurs est codée en Java et nécessite l'intervention d'une personne compétente en programmation. Ainsi, la définition de règles OpenRules est accessible aux utilisateurs métier et l'intervention de techniciens informatiques est minimisée. Le produit obtient une note de 2 sur 2 pour ce critère.

5.4.2 Abstraction des détails d'implémentation

Drools

Drools a pris soin de proposer des moyens d'abstraction des détails d'implémentation dans plusieurs modules. Pour ce qui est des tables décisionnelles, les colonnes renfermant des détails techniques peuvent être cachées. On obtient alors un affichage optimisé pour l'utilisateur métier.

5	Table de règles de calcul d'intérêt				
9	type	monnaie	solde (min,max)	mois	taux d'intérêt
10	TRANSACTIONNEL	EUR			"0.01"
11	ETUDIANT	EUR	0, 2000		"1.00"
12	EPARGNE	EUR	0, 100	0, 1	"0.00"
13	EPARGNE	EUR	0, 100	1, 3	"0.10"
14	EPARGNE	EUR	0, 100	3, 12	"2.00"
15	EPARGNE	EUR	100, 1000	0, 1	"0.10"
16	EPARGNE	EUR	100, 1000	1, 3	"3.00"
17	EPARGNE	EUR	100, 1000	3, 12	"3.25"
18	EPARGNE	EUR	1000, 5000	0, 1	"0.10"
19	EPARGNE	EUR	1000, 5000	1, 3	"3.25"

Figure 5.13 Les détails techniques de la table décisionnelle sont cachés

Par ailleurs, les DSLs abstraient le code de définition des conditions des règles et des procédures exécutables en conséquence. Les définitions des flux *Drools Flow* sont

également abstraites par un éditeur graphique de définition des processus qui génère automatiquement le code équivalent aux diagrammes. À travers ces couches d'abstraction, Drools cache largement les détails d'implémentation et mérite une note de 2 pour ce critère.

CLIPS

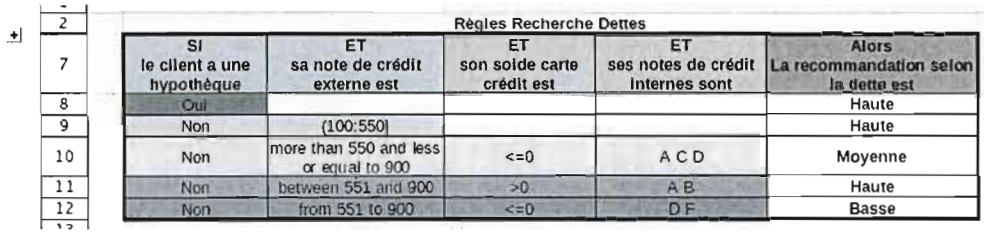
Étant destiné à s'interfacer à des solutions logicielles globales, le moteur CLIPS expose l'ensemble de ses fonctionnalités via ses commandes prédéfinies et son API C. L'objectif du projet n'est pas de fournir une couche d'abstraction au-dessus du moteur, mais uniquement le "cerveau" destiné à enregistrer des faits et exécuter des règles. Il est donc possible d'encapsuler les interactions avec le moteur en masquant les détails du langage natif, mais le produit évalué n'en fournit pas. Ainsi, une note de 0 sur 2 est attribuée pour ce critère.

OpenRules

OpenRules fait un effort important pour séparer les détails d'implémentation des données métier. En utilisant les mécanismes fournis par les tableurs, le code Java peut être entièrement caché à l'utilisateur métier.

2	Règles Recherche Dettes				
3	Rules void **reglesRechercheDette**(DemandeEmprunt emprunt, Client c)				
4	C1	C2	C4	C5	A1
5	c.possedeHypothe que.equals(YN)	interval.contains(c.note CreditExterne)	op.compare(c.solde CarteCredit.valeur)	contains(notes, c.notesInternesCredit)	emprunt.resultatRecherche Dette = niveau; out("Resultat Recherche Dette:"+niveau);
6	String YN	FromToInt interval	CompareToInt valeur	String[] notes	String niveau
7	**Si le client a une hypothèque**	**ET sa note de crédit externe est**	**ET son solde carte crédit est**	**ET ses notes de crédit internes sont**	**Alors La recommandation selon la dette est**
8	Oui				Haute
9	Non	[100;550]			Haute
10	Non	more than 550 and less or equal to 900	<=0	A C D	Moyenne
11	Non	between 551 and 900	>0	A B	Haute
12	Non	from 551 to 900	<=0	D F	Basse

Figure 5.14 Détails techniques exposés



Règles Recherche Dettes

SI le client a une hypothèque	ET sa note de crédit externe est	ET son solde carte crédit est	ET ses notes de crédit internes sont	Alors La recommandation selon la dette est
Oui				Haute
Non	(100:550]			Haute
Non	more than 550 and less or equal to 900	<=0	A C D	Moyenne
Non	between 551 and 900	>0	A B	Haute
Non	from 551 to 900	<=0	D F	Basse

Figure 5.15 Détails techniques cachés

Ce dernier pourra alors remplir des tables décisionnelles dont la signification des colonnes est exprimée en langage naturel. Comme le montre l'exemple utilisé dans les figures 5.14 et 5.15, certaines classes utilitaires OpenRules abstraient les constructions du langage Java. Les éléments d'un tableau peuvent être exprimés en les séparant par des espaces (colonne note de crédit interne de l'exemple). Les conditions de type (borne < x) (x < borne) peuvent être exprimées avec un langage naturel ou des notations mathématiques d'intervalles tel que [b1, b2) (colonne note de crédit externe). Il est également possible de comparer un entier avec une valeur numérique de type différent, en exprimant l'opérande et la valeur dans une seule cellule de la table décisionnelle (colonne solde carte crédit).

L'utilisation de tableurs, la séparation de l'implémentation Java des données métier, la possibilité de cacher l'implémentation Java et les classes utilitaires OpenRules concourent à abstraire les détails techniques de l'utilisateur non informaticien. Le produit obtient la note de 2 sur 2 pour ce critère.

5.4.3 Interfaces pour non informaticiens

Drools

L'accès aux fonctionnalités de Drools peut se faire via deux types d'interfaces : Les plugiciels de l'environnement de développement *eclipse* et l'application web Guvnor.

Les plugiciels *eclipse* offrent un assistant de définition de règles, de DSL et de flux, ainsi que des outils d'administration des dépôts de règles. Lorsqu'on lui spécifie un modèle

de données, l'assistant d'édition de règles permet à l'utilisateur de sélectionner les entités et leurs attributs via des interfaces graphiques. De plus, l'éditeur de règles propose l'auto-complétion des phrases lorsqu'un DSL est spécifié.

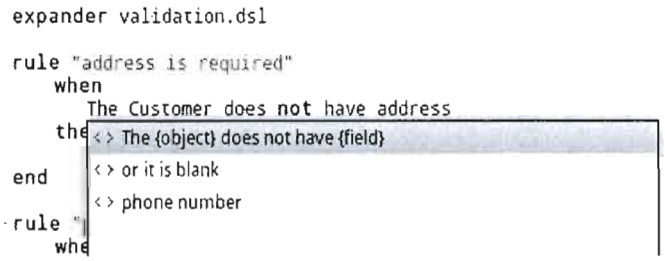


Figure 5.16 Auto-complétion des règles Drools

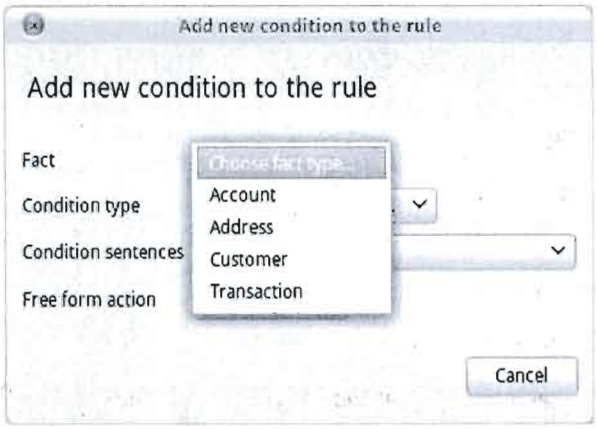


Figure 5.17 Assistant de définition de règles Drools

Par ailleurs, l'application web Guvnor reprend les fonctionnalités du plugiciel *eclipse* en y rajoutant un contrôle avancé des permissions d'accès aux bases de connaissances. L'accès à l'application se fait avec un nom d'utilisateur et un mot de passe. L'administrateur est capable de créer des utilisateurs de Guvnor et de leur affecter des droits. La figure 5.18 montre que l'utilisateur *jihedamine* possède les droits d'administration pour le paquetage *banque.emprunts*, les droits de lecture pour le paquetage *banque.modele* et les droits du profil d'analyste pour la catégorie *Hypotheques*.

En fonction des droits qu'il possède, un utilisateur est capable de parcourir et manipuler

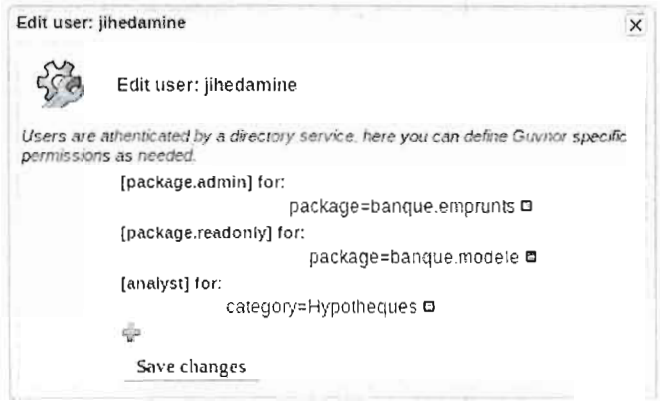


Figure 5.18 Contrôle des permissions des utilisateurs de Guvnor

les règles, les fonctions, les DSL, les modèles de données et autres éléments de la base de connaissances.

Les utilisateurs sont également encadrés pour définir des faits via des assistants qui les orientent dans les choix à faire.

Enfin, Guvnor permet de définir des tables décisionnelles à partir du navigateur web. Les conditions et les actions sont définies via des assistants. Les valeurs sont ensuite saisies dans le tableur généré.

Ainsi, Drools répond, à travers son application web Guvnor, aux exigences définies pour ce critère : un environnement d'édition puissant, ergonomique, web, muni d'assistants et imposant des restriction d'accès. Le produit obtient une note de 2 pour ce critère.

CLIPS

CLIPS fournit une interface graphique pour les systèmes d'exploitation Windows et Mac et pour l'environnement graphique X (existe pour Linux et Solaris entre autres). Les interfaces sont identiques en terme de fonctionnalités et offrent certaines fonctionnalités dont :

- Une fenêtre de parcours du système de fichiers pour sélectionner un fichier à charger dans le moteur CLIPS.

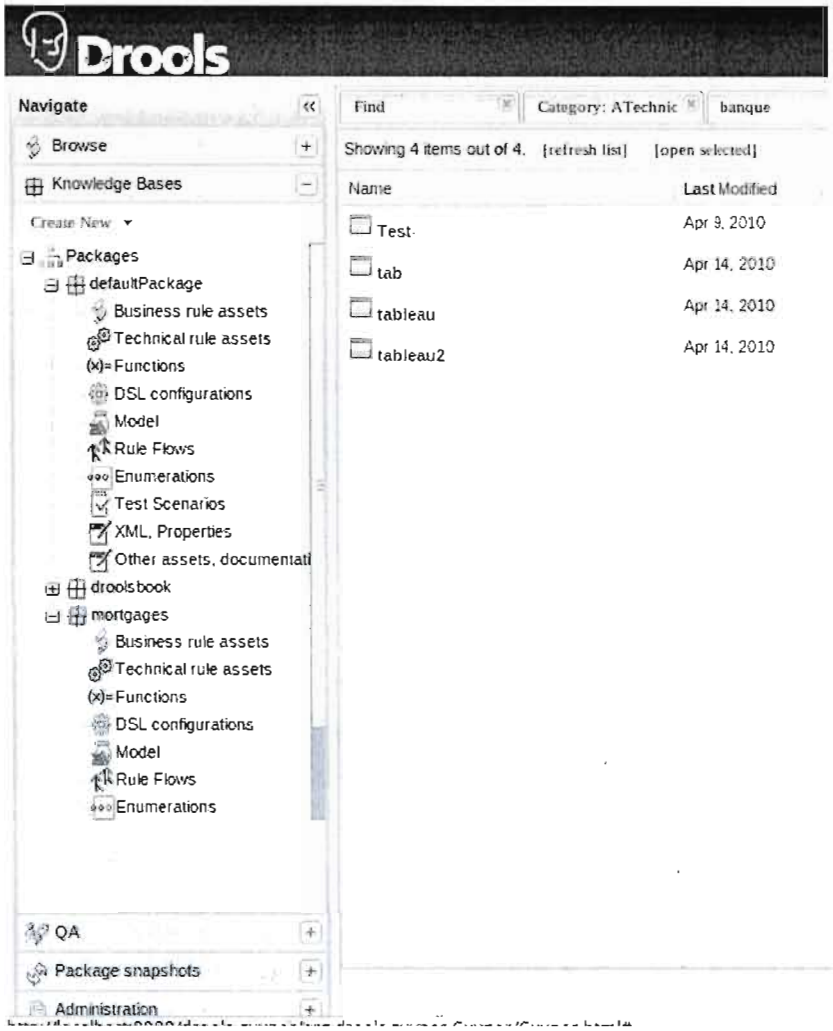


Figure 5.19 Présentation des bases de connaissances par Guvnor

- Un éditeur de texte basique comprenant la possibilité d'annuler les changements (*undo*, *redo*), de commenter une ligne, de chercher/remplacer du texte ou encore de charger le contenu de l'éditeur textuel dans la base de connaissances.
- L'affichage d'avertissements avant l'exécution de certaines commandes (Demande de confirmation avant d'effectuer un *clear*...)
- L'auto-complétion des commandes.
- L'accès aux commandes CLIPS depuis un menu graphique (*reset*, *run*, *watch*, *sa-*

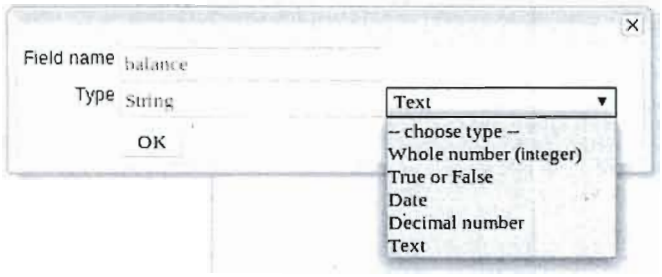


Figure 5.20 Assistant de définition de faits Guvnor

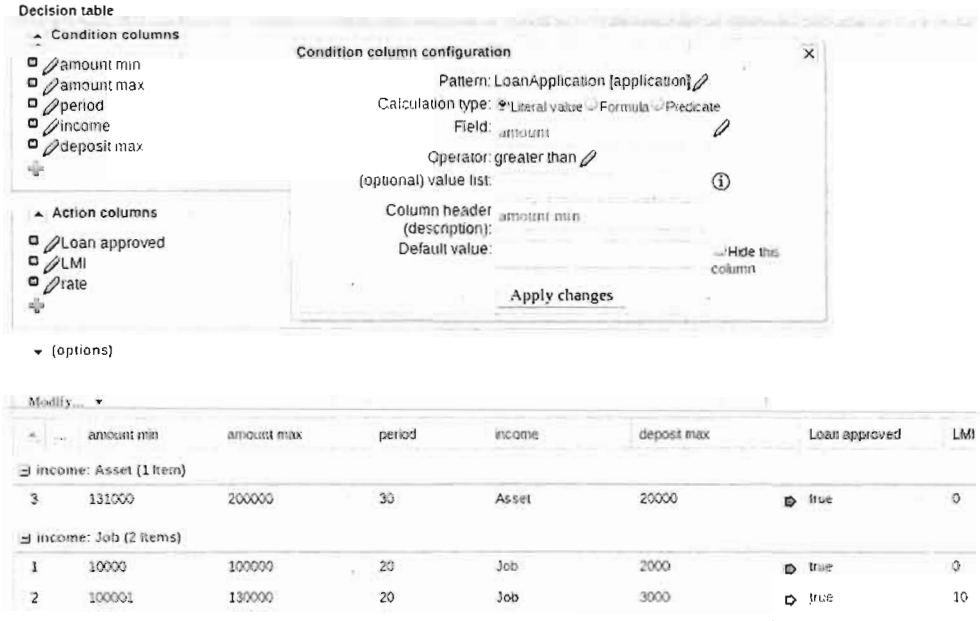


Figure 5.21 Tables décisionnelles web Guvnor

lience...) : Pour certaines de ces commandes, des assistants sont proposés avec des listes déroulantes et des cases à cocher afin de guider l'utilisateur dans ses choix.

- Parcours et gestion des modules, règles, faits, classes, fonctions, etc : Pour chaque type d'entités, une liste est affichée et il est possible de modifier une entité donnée en la sélectionnant dans la liste. Les actions applicables pour une entité dépendent de son type (afficher les *message handlers* (méthodes) d'une classe, afficher les méthodes implémentant une fonction générique, supprimer une variable globale, ajouter un

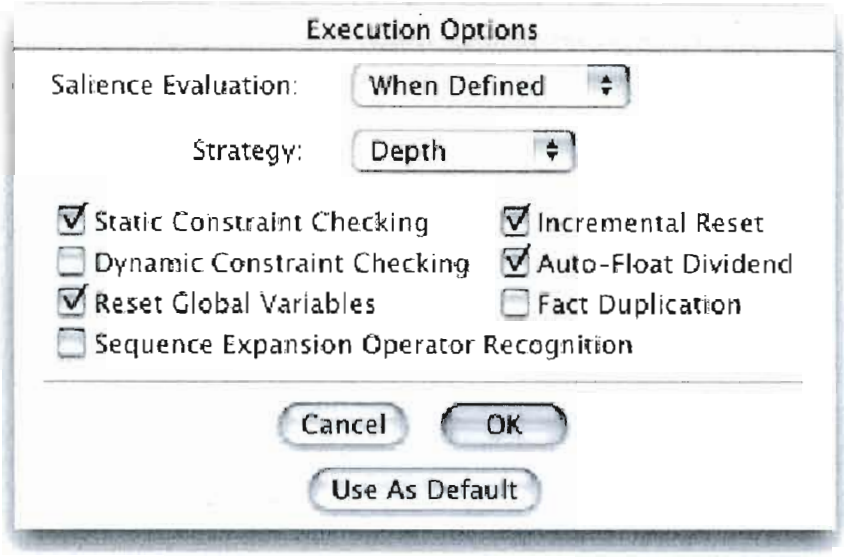


Figure 5.22 Assistant d'affectation d'une priorité à une règle

point d'arrêt dans une règle...). Toutes ces fonctionnalités représentent des commandes CLIPS réalisables à partir de la ligne de commande basique.

- Affichage de l'ensemble des commandes CLIPS correspondant à une sous-chaîne de caractères saisie par l'utilisateur : Un bouton "Aide" permet d'afficher de la documentation sur le fonctionnement d'une commande sélectionnée dans la liste.

Les assistants d'exécution de commandes, les messages d'avertissement et la possibilité de gérer graphiquement les éléments de la base de connaissances apportent une aide appréciable. Cependant, comparée à ce que propose Drools, l'interface CLIPS reste basique et n'apporte pas une grande valeur ajoutée par rapport à la ligne de commande. Le produit obtient donc une note de 1 sur 2 pour ce critère.

OpenRules

Le système fournit un plugiciel *eclipse* qui permet de créer des projets OpenRules. L'assistant propose à la fois des exemples illustratifs complets et des squelettes de plusieurs profils de projets OpenRules (projet utilisant des données XML, des classes Java, produisant un service web, etc). Les fichiers de construction et d'exécution (build.xml de

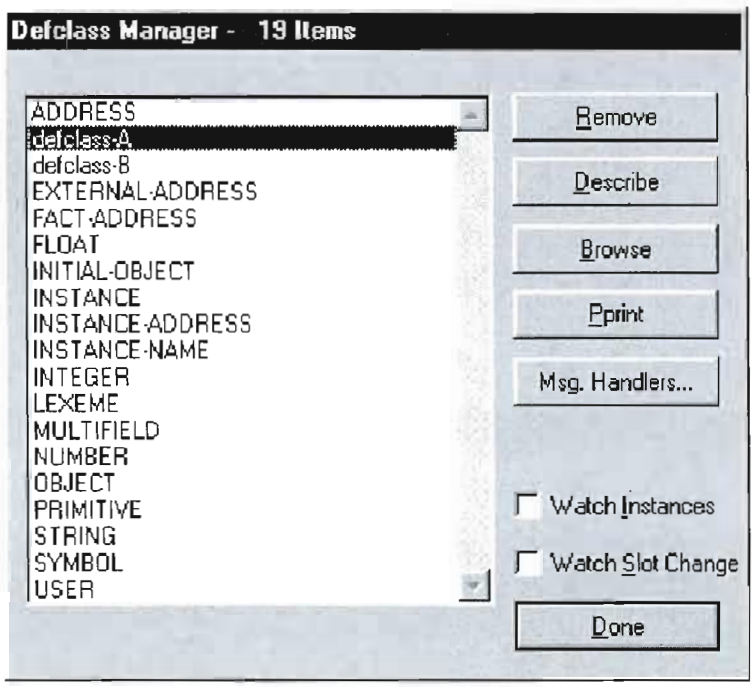


Figure 5.23 Gestion de la liste de classes

ant) sont générés selon le profil sélectionné. Si le projet utilise des formulaires web, le fichier d'accueil JSP et le fichier de configuration *web.xml* sont également générés. S'il s'agit d'un projet avec exposition des règles comme service web, les fichiers de configuration et les classes requises sont également générées à la création du projet.

De plus, le plugiciel *eclipse* détecte les erreurs dans les fichiers de tableurs pendant la conception du projet OpenRules et affiche un compte rendu avec la cause et l'emplacement des erreurs.

Par ailleurs, les tableurs tel que Excel ou OpenOffice fournissent des interfaces intuitives aux non informaticiens tout en contrôlant leur accès aux données. La figure 5.25 montre la saisie d'une valeur dans une liste déroulante. Cette fonctionnalité est facilement réalisable dans Excel ou OpenOffice et permet d'orienter l'utilisateur dans ses choix.

Enfin, l'utilisateur final du système pourra exécuter les règles via des formulaires web

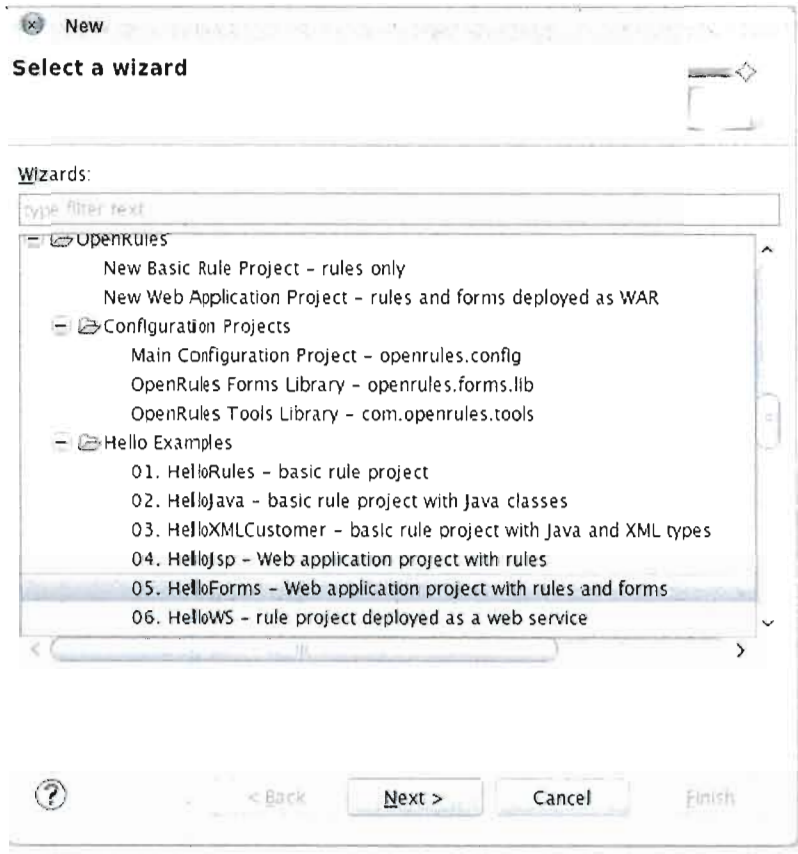


Figure 5.24 Création d'un projet OpenRules avec le plugiciel *eclipse*

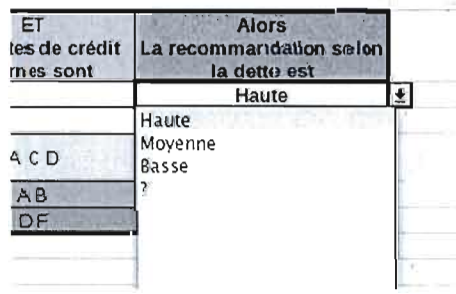


Figure 5.25 Choix d'une valeur dans une liste prédéfinie

(pages HTML + JSP (Java Server Page)), déployables dans un conteneur web tel qu'Apache Tomcat et accessibles à travers le réseau interne de l'organisme. Les formu-

laires constituent une interface ergonomique et facile d'accès, avec des menus déroulants, des cases à cocher et une navigation dirigée par les boutons de navigation (suivant, précédent).

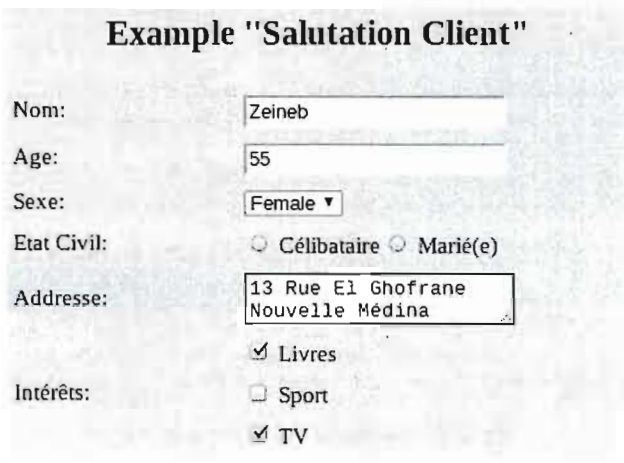


Figure 5.26 Interface web d'exécution d'une règle

OpenRules permet d'utiliser des méthodes Java ou Excel pour valider les données des formulaires. En cas d'erreurs, l'attribut dialog().errors est incrémenté et un message d'erreur est retourné par la méthode de validation. S'il n'y a aucune erreur, une chaîne de caractères vide est retournée.

Ce type de méthodes est préférablement placé dans une cellule adjacente aux données à valider, afin que les éventuels messages d'erreurs s'affichent à côté des données non valides. S'agissant de méthodes Java, les validateurs peuvent concerner plusieurs données et effectuer des vérifications sophistiquées. Leur code doit toutefois être défini par une personne compétente en programmation Java.

La validation ne se fait pas en temps réel mais après que l'utilisateur métier ait envoyé le formulaire.

Certaines méthodes de validation prédéfinies sont fournies par OpenRules, tel que *validateIntRange(int nombre, int min, int max)*, *validateStringLength(String chaine, int min, int max)* ou encore *validateEmail(String chaine)*.

Ainsi, OpenRules propose des interfaces de conception de règles pour les utilisateurs

Layout TableLayout FenetreDonneesEmprunt(DemandeEmprunt emprunt)		
header("Demande emprunt")		
"Montant emprunt:"	[emprunt.montant]	validateIntRangeLayout(emprunt.montant, 1000,1000000);
"Raison emprunt:"	[emprunt.raison]	
"Durée emprunt:"	<F type="radio"> [emprunt.duree] ["36,48,60 72"] </F>	validerDureeEmprunt(emprunt);

Figure 5.27 Validation des faits saisis

compétents techniquement (plugiciel *eclipse*) et les experts métier (tableurs) ainsi que des interfaces d'exécution des règles avec les formulaires web. Ces derniers peuvent vérifier la validité des données saisies par l'utilisateur métier et l'informer des éventuelles erreurs. Le produit obtient une note de 2 sur ce critère.

5.5 Le moteur de règles

5.5.1 Stratégies du moteur d'inférence

Drools

Le moteur d'inférence de Drools utilise une variante de l'algorithme d'appariement de modèles Rete appelée Rete OO qui adapte l'algorithme original aux structures orientées objet. Le chaînage arrière n'est pas implémenté par Drools. Par contre, le mot clé *from* permet d'appliquer des raisonnements sur des données qui ne sont pas dans la mémoire de travail. Il peut s'agir du résultat d'un appel de méthode ou d'une variable comme dans l'exemple suivant :

```
rule "valider code postal"
when
    Personne( $adressePersonne : adresse )
    Adresse( codePostal == "H2X3Y3") from $adressePersonne
then
   # le code postal est valide
end
```

Comme Drools ne permet que le raisonnement en chaînage avant, il n'obtient que 1 sur 2 pour ce critère.

CLIPS

CLIPS ne permet directement que le raisonnement en chaînage avant. Il est toutefois possible de simuler le raisonnement en chaînage arrière et plusieurs implémentations ont été réalisées.

Par exemple, Eclips (Homeier et Le, 1991) décrit un projet de l'*Aerospace Corporation*[13] qui a étendu CLIPS en introduisant la notion de modules pour réaliser le chaînage arrière. L'article note que « les systèmes de production requièrent une combinaison du chaînage avant et arrière pour refléter le processus de raisonnement d'un expert humain ». Lorsque le système de production ne fournit que le chaînage avant, les développeurs ont tendance à « utiliser des données pour réaliser des structures de contrôle guidées par les buts à atteindre ». Eclips (Homeier et Le, 1991) considère cette approche imparfaite et introduit la notion de buts. L'extension permet de regrouper les règles en modules et stocke une pile de buts qui référencent chacun les modules utilisés pour sa réalisation. Lorsqu'un but est visé, seules les règles des modules qu'il référence peuvent être exécutées. Ainsi, l'activation de buts se fait par chaînage arrière et l'exécution de règles amenant à la réalisation du but se fait par chaînage avant.

Par ailleurs, un professeur de l'Université britannique du Middlesex propose sa simulation du chaînage arrière[14], similaire à l'extension Eclips (Homeier et Le, 1991). Le professeur note que le projet est uniquement à titre illustratif et n'est pas destiné à un environnement de production. Sa réalisation n'a pas demandé un temps et un effort de développement considérable.

Ainsi, CLIPS offre directement le raisonnement en chaînage avant, mais il est aisé de simuler le raisonnement en chaînage arrière et de combiner les deux stratégies dans un

13. http ://www.aero.org/

14. http ://www.cwa.mdx.ac.uk/bis2040/backchain/backchain.html

même système expert. Le produit obtient donc une note de 2 sur 2.

OpenRules

Le moteur d'exécution de règles d'OpenRules mouline le fichier de tableur principal (spécifié dans la configuration du projet) et les éventuels fichiers de tableur qu'il inclut. Ensuite, il exécute la méthode principale spécifiée dans le tableur ou dans une classe Java (dans les deux cas, il s'agit d'instructions Java). Cette méthode invoque à son tour une séquence de règles. À l'intérieur d'une même table décisionnelle, les règles sont exécutées les unes après les autres, selon leur ordre de placement dans la table : De haut en bas pour les tables décisionnelles verticales et de gauche à droite pour les tables horizontales. Les conditions d'une règle sont également évaluées l'une à la suite de l'autre. Si elles sont toutes satisfaites, alors toutes les actions de la règle sont exécutées. Les tables décisionnelles sans type de retour (*Rules void nomDeLaRegle(parametres)*) exécutent les actions de toutes les règles dont toutes les conditions sont satisfaites. Les tables décisionnelles avec un type de retour (*Rules String nomDeLaRegle(parametres)*) retournent un résultat (spécifié dans l'action de la règle) pour la première règle dont toutes les conditions sont satisfaites ; les règles qui sont placées après ne sont pas évaluées. Il est clair que l'exécution des règles se fait selon un algorithme très naïf. Le flux d'exécution est séquentiel, monolithique et identique en tout point à l'exécution d'un ensemble d'instructions dans un code procédural classique. Les ensembles de règles sont des méthodes Java appelées les unes à la suite des autres. Les règles et leurs conditions sont parcourues selon une boucle itérative basique. Les notions d'optimisation de règles, d'appariements partiels et autres techniques utilisés par les algorithmes de chaînage avant tel que Rete sont totalement absentes.
OpenRules ne propose donc pas de moteur d'exécution de règles optimisé et se contente des constructions du langage Java. Il obtient une note de 0 sur 2 sur ce critère.

5.5.2 Explication du raisonnement et gestion de l'incertitude

Drools

Drools offre un mécanisme de maintien de la vérité qui met à jour la base de connaissances de manière automatique. Pour être maintenu par le système, un fait doit être inséré avec l'instruction *logicalInsert*. Cette instruction permet d'enlever automatiquement un objet quand la règle devient fausse.
Par exemple, une règle ajoute à un conducteur la qualité de *novice* avec un *logicalInsert*, s'il conduit depuis moins de deux ans. Dès que le conducteur dépasse les deux ans d'expérience, le fait d'être novice est automatiquement retiré.
Par ailleurs, la gestion de l'incertitude est absente de Drools. En effet, le produit ne prend pas directement en charge l'affectation de scores aux conditions. La seule façon de le réaliser est de calculer les scores dans des règles distinctes et d'avoir une règle avec une priorité inférieure qui s'exécute plus tard et décide de déclencher des actions ou pas selon les scores calculés précédemment.
Davide Sottara, un étudiant en doctorat, effectue actuellement des recherches sur l'implémentation de la gestion de l'incertitude dans Drools (Sottara, Mello et Proctor, 2008), mais ses travaux ne sont pas encore prêts à être utilisés dans un environnement de production.
Ainsi, Drools satisfait partiellement ce critère et obtient la note de 1 sur 2.

CLIPS

CLIPS réalise le maintien de la vérité avec les connexions logiques et les dépendances :

- Connexions logiques : En insérant un fait avec le mot clé *logical*, un lien est créé entre le fait et les prémisses sur lesquelles il est basé. Lorsque les prémisses changent, leurs résultats ne sont plus valides. Le fait est donc retiré de la base de faits. Le connexion logique se fait entre la condition et l'action de la règle. Si la condition n'est plus vraie, les faits qui en résultaient sont retirés.

```
(defrule reaction-fumee
        (logical (fumee-presente))
```

```
        =>
        (assert (alarme-activee))
)
```

Lorsqu'il y a de la fumée, on peut conclure que l'alarme est activée. Si le fait *fumee-presente* est retiré, *alarme-activee* le sera également.

- Dépendances : La fonction *(dependencies <identifiant-entite>)* liste les appariements partiels à partir desquels l'entité (fait ou instance) passée en paramètre a été déduite par connexions logiques.

Example

```
(defrule lister-dependences
   ?f <- (fait $?)
   =>
   (dependencies ?f))
```

L'identifiant de l'entité peut être une variable initialisée dans les conditions de la règle, un indice de fait (par exemple 3 pour le fait nommé f-3), ou un nom d'instance (par exemple [objet1]).

La fonction *(dependents <identifiant-entite>)* liste toutes les entités qui sont conclues à partir de l'entité passée en paramètre. En reprenant l'exemple du détecteur de fumée, on aurait :

```
CLIPS> (assert (fumee-presente))
CLIPS> (run)
CLIPS> (facts)
f-0     (fumee-presente)
f-1     (alarme-activee)
CLIPS> (dependents 0)
f-1
CLIPS> (dependents 1)
None
```

Concernant la gestion de l'incertitude, une extension nommé FuzzyCLIPS a été développée par l'institut des technologies de l'information du CNRC (Conseil National de Recherche

du Canada) [15]. Le projet n'est plus maintenu depuis une dizaine d'années mais son code source a été republié sur le web par des utilisateurs de CLIPS [16].
FuzzyCLIPS rajoute les concepts d'incertitude et de logique floue à CLIPS. Il s'intègre avec les faits et le moteur d'inférence, permettant ainsi de facilement manipuler des faits et des règles floues. FuzzyCLIPS peut gérer des faits exacts, flous ou une combinaison des deux approches. Sa maintenance n'étant plus assurée par le CNRC, sa mise à jour par rapport aux nouvelles versions de CLIPS nécessite un effort significatif (recompilation, compatibilité des bibliothèques C...).
Le maintient de la vérité est donc géré directement par CLIPS. La gestion de l'incertitude et la logique floue sont réalisables en étendant le système. Cependant, le fait que l'extension FuzzyCLIPS ne soit plus maintenue nous empêche d'accorder la note de 2 sur 2 sur ce critère. CLIPS obtient donc une note de 1 sur 2.

OpenRules

Étant donné que le moteur d'OpenRules ne fait qu'exécuter une méthode principale, qui à son tour invoque d'autres méthodes Java "classiques", le système ne maintient pas une séquence de dépendances entre l'exécution de règles et la production de faits. La logique floue ne bénéficie pas non plus d'une prise en charge directe par le système OpenRules. Le produit ne satisfait aucunement ce critère et obtient une note de 0.

5.5.3 Intégration à l'environnement

Drools

Le moteur de règles et les composants de base de Drools étant des bibliothèques Java, le système est multi-plateformes.
L'intégration avec les autres composants de l'environnement est facilitée par la présence

15. http ://www.nrc-cnrc.gc.ca/eng/ibp/iit/research/knowledge-discovery.html

16. http ://github.com/cygal/FuzzyCLIPS

d'une application nommée *drools-server* déployable sur serveur d'application. Lorsque l'application reçoit une requête client, elle crée une session Drools, définit les variables globales, insère les faits et exécute les règles. À la fin de l'exécution, les résultats sont retournés au client. L'application est accessible via des interfaces SOAP et REST. Les données peuvent être échangées sous format XML ou JSON. Avec une telle architecture, il est possible de faire interagir le moteur de règles avec des applications écrites dans différents langages et s'exécutant sur des plate-formes diverses.

L'interface Guvnor livrée avec Drools est une application web codée en GWT. Il s'agit d'une interface sophistiquée et enrichie de plusieurs fonctionnalités. Il est cependant possible d'utiliser n'importe quelle autre couche de présentation : une application Java en ligne de commande, une interface Swing, une application web JSP... Ainsi, un organisme pourrait accéder aux fonctionnalités du moteur de règles Drools en étendant l'application de bureau ou l'application web qu'il utilise déjà pour d'autres systèmes.

Les applications web fournies avec Drools sont testées avec les serveurs d'applications JBoss Application Server. Guvnor peut être téléchargé dans un paquetage contenant une version optimisée de JBoss AS. Le déploiement sur le conteneur web léger Apache Tomcat est également expliqué dans la documentation officielle du produit.

Au niveau de la couche de données, il est possible d'enregistrer et d'extraire les données sous n'importe quel format (XML, fichier plat, CVS (valeurs séparées par des virgules)...) via des cadres Java tel que iBatis et Hibernate.

Enfin, les données provenant d'applications tierces peuvent être intégrées aux définitions des règles en utilisant le mot clé *from* qui permet d'appliquer les règles sur des données récupérées, par exemple, d'une base de données relationnelle via Hibernate.

Ainsi Drools offre des points d'intégration aux niveaux couche de présentation, couche métier et couche de données. Il obtient la note complète sur ce critère.

CLIPS

CLIPS est multi-platesformes et exécutable sur les systèmes d'exploitations majeurs (Windows, Mac, *Nix).

La commande *(system)* de CLIPS réutilise la fonction *system* fournie par le langage C pour la plupart des systèmes Unix. L'exemple suivant montre l'appel de la commande système pour afficher le contenu d'un répertoire :

```
(defrule print-directory
   (print-directory ?directory)
   =>
   (system "dir " ?directory))
```

En plus de cette intégration bas niveau, CLIPS peut être incorporé à des applications tierces. On peut également définir des fonctions dans des langages externes et les appeler depuis CLIPS. En plus de la prise en charge directe du langage C, plusieurs autres langages possèdent une bibliothèque d'interfaçage avec CLIPS, dont Python (PyCLIPS), Java (CLIPSJNI), ADA...

L'utilisation de CLIPS comme composant embarqué est utile lorsque le moteur de règles constitue une petite partie du système informatique ou qu'on a besoin de partager des données avec d'autres fonctions. Dans ce cas, CLIPS peut être appelé comme sous-routine à l'intérieur du code applicatif du système d'information. Le code suivant montre l'utilisation de CLIPS à l'intérieur de code C.

```
main()
      {
      // la variable uneClasse representant une classe CLIPS
      // peut etre utilisee ailleurs dans l'application
      void *uneClasse;
      InitializeEnvironment();
      Reset(); // equivaut a la commande (reset) saisie dans CLIPS
      uneClasse = FindDefclass("USER");
      AssertString("(compte a-un-solde-de 1340)"); // equivaut a (assert
          (compte a-un-solde-de 1340))
      Run(-1L) // equivaut a (run)
      }
```

Il peut aussi être utile d'appeler des fonctions externes à partir de CLIPS. Les variables et les valeurs littérales peuvent être envoyées à des fonctions externes qui retournent

des résultats au moteur de règles. Un exemple Python illustrant cette fonctionnalité a été donné lors de l'évaluation du critère *5.3.2 Procédures et algorithmes.*

La facilité d'ajouter des fonctions externes à CLIPS permet de l'étendre et de le personnaliser à volonté. Il est alors possible d'utiliser différentes implémentations de la couche présentation, d'extraire des données de sources externes diverses, etc. Plusieurs extensions de CLIPS ont d'ailleurs vu le jour, et ce dans des environnements très variés, allant du domaine médical à l'aérospatial en passant par le secteur bancaire.

Toutefois, la communication entre CLIPS et des langages autres que C peut poser des problèmes comme la transformation des types de données ou la gestion de la mémoire. La transformation des entités Java ou Python vers CLIPS doit entièrement être effectuée par l'utilisateur. Cette tâche pourrait s'avérer ardue pour un organisme de finance sociale et solidaire qui possède un personnel informatique limité.

Ainsi, même si CLIPS est très extensible et peut être intégré à une large panoplie d'environnements, la procédure exige un niveau technique et un effort de développement considérables. CLIPS est essentiellement un moteur d'inférence et non un système de gestion de règles fournissant des abstractions de haut niveau. Il obtient une note de 1 sur 2 pour ce critère.

OpenRules

L'utilisation d'OpenRules requiert l'environnement Java et une application de tableurs. Java est multi-plateformes et les applications de tableurs existent sur les systèmes d'exploitation majeurs : si Excel n'est disponible que sous Windows et Mac, le tableur OpenOffice est également disponible sous Linux et Solaris. Ainsi, OpenRules est accessible quelque soit le système d'exploitation de l'organisme client.

L'API *OpenRulesEngine* permet d'invoquer le moteur de règles depuis une application Java. Le moteur exécute les règles d'affaire et retourne les résultats dans des objets Java passés en paramètres au moteur. Ainsi, OpenRules permet d'intégrer les règles d'affaires présentées sous format Excel à une application Java, qu'il s'agisse d'une application autonome (standalone) ou d'une application web (JSP, par exemple).

Le système OpenRules peut aussi être publié comme service web et s'intégrer à une architecture orientée-services. Toute application prenant en charge les services web pourrait alors y accéder, quelque soit le langage de programmation utilisé.
Différentes implémentations de la couche de présentation peuvent s'interfacer avec OpenRules. La solution par défaut étant les formulaires HTML et JSP générés à partir de tableurs. Une interface codée en GWT (Google Web Toolkit) ou tout autre cadre de développement d'applications web Java pourrait s'interfacer avec le moteur. La publication d'OpenRules comme service web élargit les possibilités à des interfaces WinForms (.NET) entre autres.
Concernant l'accès aux données externes, OpenRules prend directement en charge les données déclarées en XML, dans des tables décisionnelles ou en Java. Le format des tables décisionnelles est assez répandu dans les entreprises comme moyen de modéliser les données. De plus, les bibliothèques Java tel que Hibernate et iBatis permettent d'accéder à une multitude de sources de données externes. Ainsi, OpenRules se base sur Java, les tables décisionnelles et, à moindre degré, XML ; qui sont trois supports largement répandus dans les systèmes informatiques des entreprises. Ceci assure une intégration aisée aux architectures existantes. Le produit obtient une note de 2 pour ce critère.

5.5.4 Changements dynamiques

Drools

Drools propose un agent qui permet d'ajouter des éléments à la base de connaissances pendant l'exécution du moteur de règles. L'agent vérifie périodiquement si une ressource a été modifiée. Cette dernière peut être un fichier .drl (règles exprimées avec le langage Drools), .dsl (définition d'un DSL), .dslr (règles exprimées avec un DSL), .drf (flux de règles), un répertoire contenant plusieurs ressources, etc.
Lorsqu'un changement des fichiers chargés est détecté, l'agent met à jour la base de connaissances et notifie l'ensemble des composants qui l'observent (listeners).

Le bout de code suivant montre l'instanciation d'un tel agent :

```
ResourceFactory.getResourceChangeScannerService().start();
ResourceFactory.getResourceChangeNotifierService().start();
KnowledgeAgentConfiguration conf = KnowledgeAgentFactory
  .newKnowledgeAgentConfiguration();
conf.setProperty("drools.agent.scanDirectories", "true");
final KnowledgeAgent agent = KnowledgeAgentFactory
  .newKnowledgeAgent("validation agent", knowledgeBase, conf);
```

Drools permet de retirer une règle et de la remplacer par une autre pendant l'exécution, mais ne permet pas de modifier une règle individuelle dynamiquement. Le produit obtient donc une note de 1 sur 2 pour ce critère.

CLIPS

CLIPS permet d'ajouter tout type d'entité à la base de connaissances pendant l'exécution du système. Il permet également de manipuler les règles existantes et de paramétrer leur interaction avec les faits.

- Une règle est modifiée en retirant l'ancienne version et en insérant une nouvelle. *(undefrule <nom-de-la-regle>)* supprime une règle si elle n'est pas actuellement en cours d'exécution. *(undefrule *)* permet de retirer toutes les règles. Des fonctionnalités similaires existent pour toutes les autres entités de la base de connaissances (*undefclass*, *undeffacts*, *undeffunction*, *undefgeneric*, *undefmethod*...)
- Il est possible de placer toutes les activations d'une règle donnée dans l'agenda avec la commande *(refresh <nom-de-la-regle>)*. Une activation est une combinaison de faits pour lesquelles une règle est satisfaite.
- *(set-incremental-reset <expression-booleenne>)* active ou désactive le comportement de "réinitialisation incrémentale". Lorsque le comportement est activé, les règles nouvellement définies sont mises à jour selon l'état actuel de la liste de faits. Lorsqu'il est désactivé, les règles nouvellement définies ne sont mises à jour que par les faits insérés après la définition des règles. Afin de préserver la cohérence du système, ce comportement ne peut être modifié que lorsqu'aucune règle n'existe dans la base de

connaissance.
CLIPS obtient une note de 2 sur 2 pour ses capacités de changement dynamique des règles.

OpenRules

Avant chaque exécution, le moteur de règles d'OpenRules vérifie si le fichier de tables décisionnelles principal a été modifié. La vérification se fait d'après la dernière date de modification du fichier. Si elle est différente de celle de la dernière exécution, le moteur se réinitialise automatiquement et recharge le document ainsi que les fichiers de type tableur qu'il inclut. La fonctionnalité de mise à jour d'OpenRules est donc assez basique. Contrairement à Drools et CLIPS, il n'est pas possible de rétracter ou d'insérer individuellement des règles dans la base de connaissances. Le moulinage des tableurs, avec toutes les tables décisionnelles qu'ils contiennent est repris à zéro à chaque mise à jour. On note également que seul le fichier principal est vérifié. Si des documents invoqués par ce fichier ont été mis à jour, le changement n'est pas détecté. De plus, les changements ne sont pas effectifs en cours d'exécution, mais au déclenchement de la prochaine exécution. Le produit obtient donc une note de 0 sur 2 car il ne prend pas en charge la modification de règles en cours d'exécution.

5.6 Outils de vérification des règles

5.6.1 Vérification de la syntaxe des règles

Drools

Le plugiciel *eclipse* de Drools offre la coloration syntaxique en temps réel des règles et permet ainsi de mieux visualiser leur structure.
Drools vérifie également les fichiers de tables décisionnelles (.xls, .ods...) et affiche une explication détaillée des erreurs éventuelles ainsi que leur emplacement dans la table décisionnelle. La vérification se fait en temps réel pour toutes les tables décisionnelles

```
-rule "regle simple"

    when
        $personne1 : Personne()
        $personne2 : Personne ( age > $personne1.getAge() );
    then
        System.out.println($personne2.getNom() + " est plus age que " +

end
```

Figure 5.28 Vérificateur syntaxique Drools

d'un projet Drools.

13 errors, 0 warnings, 0 others

Description	Resource	Path	Location
Errors (13 items)			
BuildError: Unable to resolve ObjectType 'Messages'	Sample.xls	/droulito/src/main/rules	line 8
BuildError: Unable to resolve ObjectType 'Messages'	Sample.xls	/droulito/src/main/rules	line 19
ImportError: Error importing: 'com.sample.DecisionTbleTest.Message'	Sample.xls	/droulito/src/main/rules	Unknown
ImportError: Error importing: 'com.sample.DecisionTbleTest.Message'	Sample.xls	/droulito/src/main/rules	Unknown
m cannot be resolved	Sample.xls	/droulito/src/main/rules	line 5

Figure 5.29 Détection des erreurs dans les tableurs

La note de 2 sur 2 est accordée sur ce critère.

CLIPS

La ligne de commande CLIPS et les interfaces graphiques fournies avec le produit vérifient les règles lorsque l'utilisateur valide leur saisie en appuyant sur entrée. Les éventuelles erreurs de syntaxe sont affichées avec des messages clairs indiquant leurs causes. Par contre, aucune coloration syntaxique en temps réel n'est fournie avec CLIPS. La fonctionnalité est toutefois disponible dans des éditeurs tiers tel que vim, qui contient un plugiciel de coloration syntaxique [17] constamment mis à jour. TextPad est un autre éditeur de texte ayant un plugiciel de coloration syntaxique pour CLIPS [18]. Contraire-

17. http ://www.vim.org/scripts/script.php ?script_id=2660

18. http ://www.textpad.com/add-ons/syna2g.html

ment à VIM, TextPad n'est disponible que pour le système d'exploitation Windows.
La note de 2 sur 2 est accordée au produit pour les fonctionnalités de coloration syntaxique.

OpenRules

Le code Java saisi dans les tableurs ne bénéficie pas de la coloration syntaxique. En revanche, il existe un plugiciel *eclipse* qui vérifie la validité des tableurs sur demande de l'utilisateur (à l'enregistrement du projet OpenRules dans *eclipse*, à l'exécution de la cible *clean* du fichier build.xml...). Après vérification, la cause de chaque erreur est expliquée et un lien permet d'ouvrir le fichier correspondant, à la cellule précise de la table décisionnelle où l'erreur s'est produite.

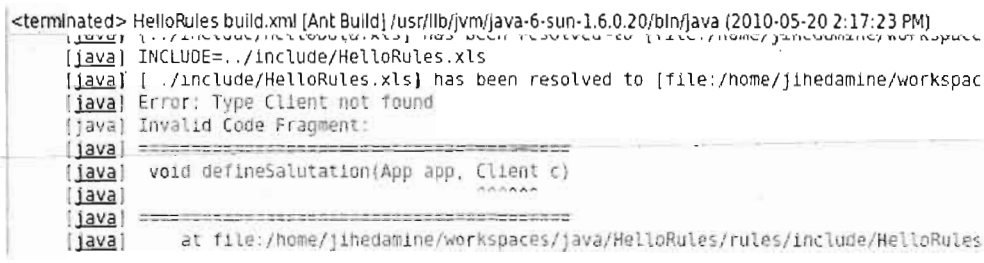


Figure 5.30 Détection d'erreurs dans les tables décisionnelles

La validation des tableurs est livrée dans une archive Java nommée "com.openrules.java.eval.jar". Elle est donc accessible en dehors d'*eclipse*.
Le code java saisi en dehors des tables décisionnelles peut être vérifié par une panoplie d'éditeurs Java, notamment *eclipse*, qui fournit la coloration syntaxique Java en temps réel.
Ainsi, OpenRules satisfait partiellement le critère car son support principal de définition de règles (i.e. les tableurs) n'offre pas la coloration syntaxique. Il faut alors passer par un plugiciel *eclipse* ou écrire le code en dehors des tables décisionnelles. Le produit obtient une note de 1 sur 2 sur ce critère.

5.6.2 Stratégies de résolution de conflit

Drools

La priorité (*salience*) et les files LIFO (dernier arrivé, premier servi) sont les deux stratégies de résolution de conflit employées par défaut. L'utilisateur peut indiquer qu'une règle possède une priorité supérieure aux autres en lui affectant une valeur de l'attribut *salience* supérieure. En cas de conflits, la règle avec la plus grande priorité est toujours choisie. Quant à la stratégie LIFO, elle utilise les valeurs de compteurs assignées dans la mémoire de travail.

Plusieurs stratégies prédéfinies sont fournies par Drools :

- Profondeur (stratégie par défaut) : Les règles les plus récemment activées sont placées avant les autres règles de même priorité.
- Simplicité : Les règles les plus récemment activées sont placées avant les autres règles de même priorité et avec une spécificité supérieure ou égale. La spécificité indique le nombre de comparaisons effectuées dans la condition de la règle (fonctions de comparaison tel que <>, comparaison avec une constante...).
- Complexité : Les règles les plus récemment activées sont placées avant les autres règles de même priorité et avec une spécificité inférieure ou égale.
- Aléatoire : Un numéro aléatoire est assigné à chaque règle activée. Ce numéro est utilisé pour ordonnancer les règles de même priorité.
- FIFO, LIFO, ordre de chargement, faits les plus récents...

La méthode *setConflictResolver* et la propriété système *drools.conflictResolver* permettent de définir des stratégies de résolution de conflit personnalisées.

Ainsi, Drools résout les conflits de règles à l'exécution en sélectionnant une règle dans l'ensemble de conflits selon une stratégie de résolution donnée. Il permet une gestion avancée des stratégies de résolution de conflits et fournit une large sélection de stratégies prédéfinies. Il obtient donc la note de 2 pour ce critère.

CLIPS

Les règles éligibles à être exécutées sont activées dans l'agenda et ordonnancées selon leur priorité. La résolution de conflit intervient pour agencer les activations de règles ayant une même priorité (même valeur de l'attribut *salience*). La commande *(set-strategy)* permet de modifier la stratégie en cours et réordonne l'agenda conformément à la stratégie sélectionnée. CLIPS fournit sept stratégies de résolution de conflit :

- Largeur : Les règles les plus récemment activées sont placées après les autres règles de même priorité.
- LEX : Ré-implémente la stratégie LEX du langage OPS5. La règle activée par les faits les plus récents est placée avant les autres règles de même priorité.
- MEA : Ré-implémente la stratégie MEA du langage OPS5. La règle dont le premier fait utilisé dans la condition est le plus récent est placée avant les autres règles de même priorité. S'il existe des règles activées avec un même premier fait dans leurs conditions, le tri entre ces règles se fait selon la stratégie LEX.
- Profondeur, simplicité, complexité, aléatoire...

Ainsi, CLIPS fournit plusieurs stratégies de résolution de conflit et permet de changer de stratégie pendant l'exécution. Toutefois, il n'est pas possible de modifier l'implémentation des stratégies existantes ou d'en déclarer de nouvelles avec des commandes du langage CLIPS. La personnalisation des stratégies n'est faisable qu'en étendant le langage. Vu cette limitation, le produit obtient la note de 1 sur 2.

OpenRules

Comme expliqué au critère *5.5.1 Stratégies du moteur d'inférence*, OpenRules n'utilise pas un algorithme de chaînage sophistiqué pour exécuter les règles, mais les invoque les unes à la suite des autres selon une approche procédurale classique. Les notions d'agenda et d'activation de règles, et par conséquent de résolution de conflits n'existent pas. Le produit obtient une note de 0 pour ce critère.

5.6.3 Débogage

Drools

Le plugiciel *eclipse* de Drools permet de définir des points d'arrêt dans les fichiers de définition de règles. Si on lance l'application en mode *débogage Drools*, l'exécution est suspendue à chaque fois qu'un point d'arrêt est rencontré dans une règle ou dans du code Java. Le fichier contenant le point d'arrêt est alors ouvert et la ligne de code active est surlignée. Les variables connues à ce stade de l'exécution peuvent être inspectées et les actions de débogage habituelles (enjamber, continuer..) peuvent être exécutées.

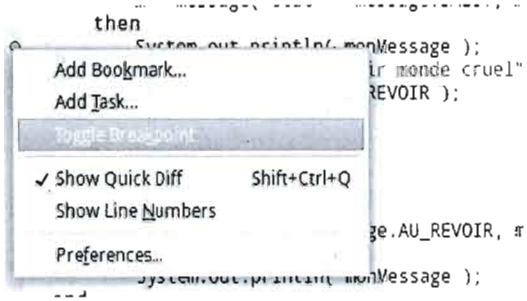


Figure 5.31 Ajout d'un point d'arrêt dans une règle

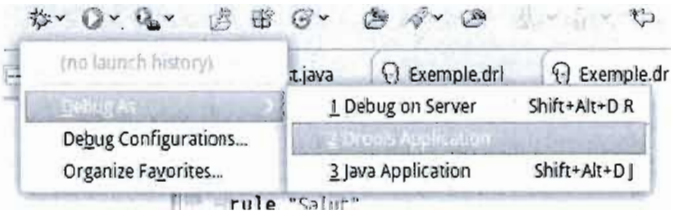


Figure 5.32 Débogage Drools d'une application

Il est également possible de déboguer les processus *Drools Flow*. Les points d'arrêt peuvent être définis pour suspendre l'exécution du flux à certaines étapes. À l'exécution, l'éditeur graphique de processus représente l'étape à laquelle le flux est rendu.
Plusieurs "vues" (fenêtres) *eclipse* peuvent aider au débogage des flux Drools :
- Une vue qui affiche le contenu de la mémoire de travail.

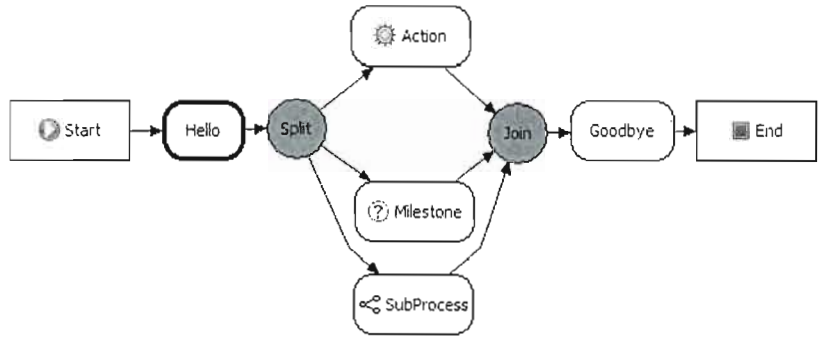


Figure 5.33 Représentation de l'activité en cours dans un flux Drools

- Une vue qui affiche les règles activées pour exécution dans l'agenda.
- Une vue des données globales.
- Les vues de débogage Java classiques qui affichent les contenus des variables définies dans le code Java et dans les règles.
- Une vue affichant les processus *Drools Flow* en exécution.
- Une vue affichant le journal d'audit.

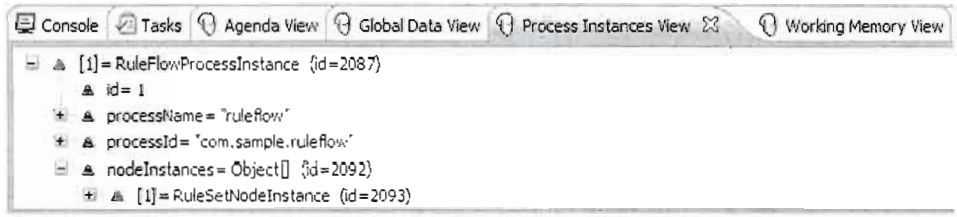


Figure 5.34 Vue *eclipse* des instances de processus Drools

Ainsi, en plus du débogage du code Java, Drools permet celui des règles et des flux. Le produit offre plusieurs vues *eclipse* affichant des informations exhaustives sur les règles et les processus en cours d'exécution. La note obtenue pour ce critère est 2.

CLIPS

La commande *(watch <elements-a-suivre>)* affiche en temps réel l'état de toute sorte d'entités de la base de connaissances (règles, faits, instances, messages, méthodes...). Par exemple *(watch facts)* affiche toutes les insertions et les rétractions de faits. Il est possible de restreindre la fonctionnalité aux faits issus d'un modèle de faits spécifique.

```
CLIPS> (reset)
<== f-0     (client-est alain)
==> f-0     (initial-fact) ; ==> signifie que le fait entre en memoire
CLIPS> (assert (client-est alain))
==> f-1     (client-est alain)
<Fact-1>
CLIPS> (retract 1)
<== f-1     (client-est alain) ; <== indique que le fait est retir\'e de
    la memoire.
```

De façon similaire, *(watch rules)* et *(watch activations)* affichent respectivement les exécutions et les (dés)activations de règles. Il est possible de restreindre le suivi à des définitions de règles spécifiques. De même, *(watch globals)* fait le suivi des variables globales, *(watch deffunctions)* celui des fonctions et *(watch instances)* celui des instances de classes. *(watch all)* active tous les types de suivi.

Par ailleurs, la commande *(run)* peut prendre un paramètre optionnel indiquant le nombre de règles à exécuter. Par exemple *(run 21)* indique à CLIPS d'exécuter 21 règles puis de suspendre l'exécution. *(run 1)* permet de faire du débogage en pas à pas (une règle à la fois). *(step)* est équivalent à *(run 1)*.

Enfin, il est possible de définir des points d'arrêt avec la commande *(set-break <nom-de-la-regle>)*. *(remove-break <nom-de-la-regle>)* fait que CLIPS ne suspend plus son exécution avant d'exécuter la règle spécifiée en paramètre. *(show-breaks)* affiche la liste des règles ayant un point d'arrêt.

```
(defrule regle1
=>
        (assert (lancer regle2))
```

```
)
(defrule regle2
=>
        (assert (lancer regle3))
)

CLIPS> (set-break regle2)
CLIPS> (reset)
CLIPS> (run)
Breaking on rule regle2
```

Ainsi, CLIPS offre le suivi paramétrable et en temps réel des entités de la base de connaissances, la suspension après l'exécution d'un nombre donné de règles ainsi que la définition de points d'arrêts. Il obtient la note de 2 sur 2 pour ce critère.

OpenRules

OpenRules n'offre aucun mécanisme de débogage et il n'est pas possible de définir des points d'arrêt ou d'effectuer une exécution pas à pas des instructions Java définies dans les tableurs. Il est toutefois possible de déboguer le code Java saisi dans un IDE (Integrated Development Environment), mais ceci est vrai pour tout code Java et ne constitue pas une valeur ajoutée d'OpenRules. Le produit obtient tout de même une note de 1 sur 2 pour ce critère, car le fait d'utiliser du code Java pour définir les règles permet de profiter des outils de débogage fournis par les IDEs, nombreux et sophistiqués, du langage.

5.6.4 Test des règles

Drools

Les développeurs peuvent tester les règles Drools en utilisant un IDE et en écrivant leurs tests unitaires en Java. Le bout de code suivant montre la méthode d'initialisation d'une classe de test JUnit ainsi qu'un test unitaire qui vérifie le bon fonctionnement d'une

règle de normalisation des adresses. Un développeur familier avec les tests unitaires Java n'aurait donc aucun apprentissage supplémentaire à faire pour tester les règles Drools.

```
@BeforeClass
public static void setUpClass() throws Exception {
 // Ajouter le fichier contenant les regles a tester a la base de
     connaissances
 knowledgeBase = DroolsHelper.createKnowledgeBase("transformation.drl");
 // Initialiser une session Drools
 session = knowledgeBase.newStatelessKnowledgeSession();
...
}

// Tester la regle qui normalise la notation des etats-unis
@Test
public void normalizationAdresseUSA() throws Exception {
  Map addressMap = new HashMap();
  addressMap.put("_type_", "Adresse");
  // Noter U.S.A pour les etats-unis
  addressMap.put("pays", "Etats Unis");
  // Executer la regle de normalisation pour les donnees de test
  execute(Arrays.asList(addressMap), "normalisationAdresseUSA");
  // Verifier que la notation U.S.A a ete convertie au format standard USA
  assertEquals("USA", addressMap.get("country"));
}
```

Drools offre également un module de test destiné aux utilisateurs métier ayant des aptitudes informatiques. Le module fait partie de l'application web Guvnor et permet de facilement écrire et maintenir des tests.

Un scénario Guvnor correspond généralement à une méthode de test JUnit. Chaque scénario définit des attentes et vérifie qu'elles sont satisfaites. Lorsqu'il est exécuté, le test crée des faits à partir de données qui lui sont passées, les insère dans la base de connaissances, exécute les règles puis vérifie les attentes. Toutes les violations sont rapportées dans l'interface Guvnor.

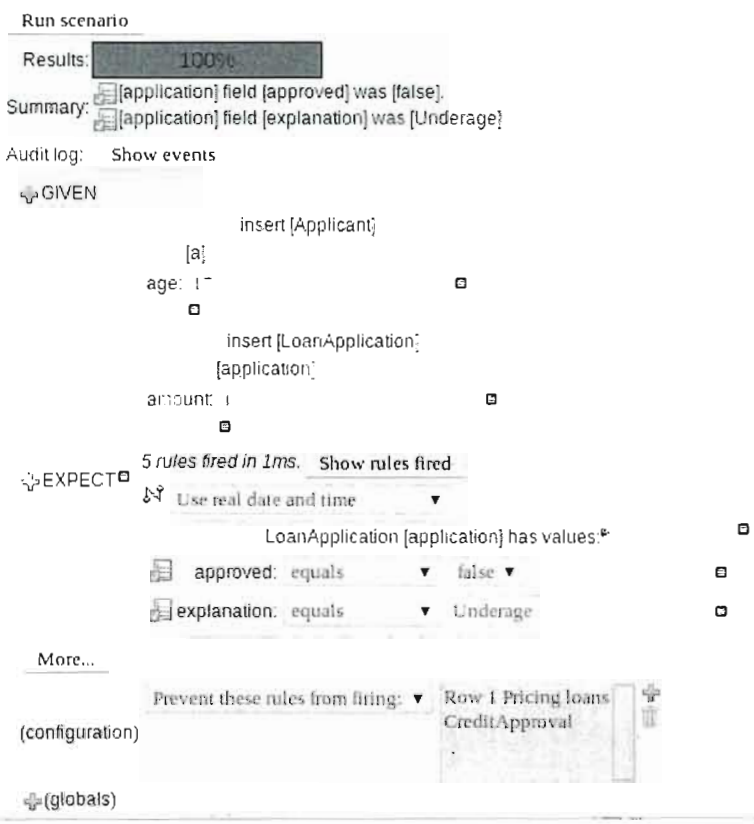


Figure 5.35 Scénario de test dans Guvnor

- La section *GIVEN* définit les faits qui seront insérés dans la base de connaissances.
- La section *EXPECT* définit les attentes. Une attente pourrait vérifier les valeurs des données, mais aussi les règles qui ont été exécutées et leur nombre d'exécutions.
- La section *Summary* liste les attentes satisfaites et non satisfaites.
- La section *Audit log* affiche une vue détaillée des événements survenus pendant l'exécution du test.

L'ensemble de règles pouvant être démarrées peut être restreint. Ceci est utile si certaines règles ont des effets collatéraux (si, par exemple, elles accèdent à des services externes).

Comme pour les suites de tests JUnit, des ensembles de tests peuvent être lancés simul-

tanément. La barre jaune indique la couverture de test, qui correspond au pourcentage de règles testées par la suite.

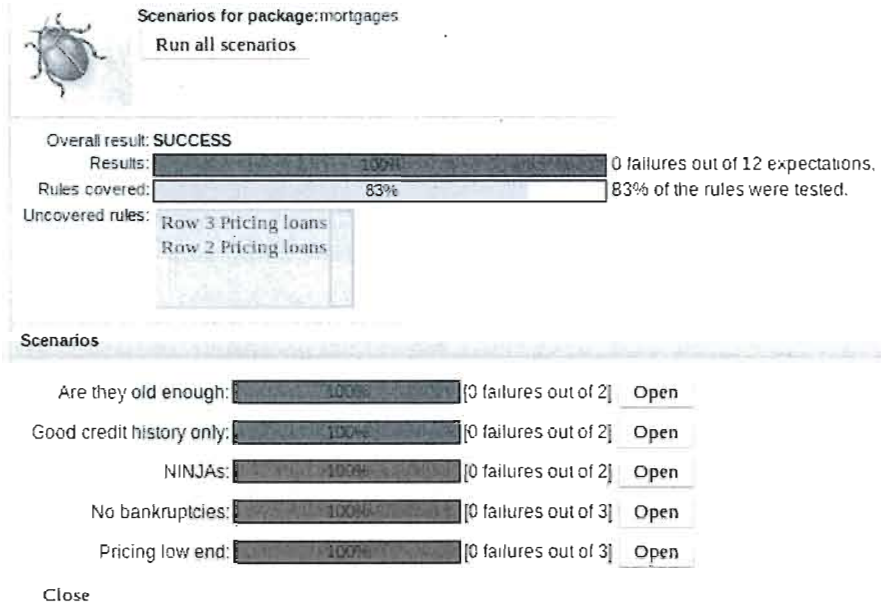


Figure 5.36 Suite de tests dans Guvnor

Enfin, les tests peuvent être invoqués en dehors de l'application Guvnor, via une URL. Ceci est utile dans le cadre de serveurs d'intégration continue qui peuvent alors lancer les tests à chaque nouvelle modification.

Drools offre donc la possibilité aux développeurs Java, autant qu'aux utilisateurs métier de définir des tests et des suites de tests unitaires. Il obtient la note de 2 pour ce critère.

CLIPS

CLIPS fournit plusieurs fonctions, appelées prédicats, qui vérifient un aspect donné pour des valeurs passées en paramètres. Par exemple, les fonctions *(numberp <expression>)*, *(floatp <expression>)*, *(oddp <expression>)* et *(stringp <expression>)* vérifient respectivement que l'expression passée en paramètre est un nombre, un réel, un nombre impair ou une chaîne de caractères. D'autres comparateurs vérifient l'égalité entre plusieurs ex-

pressions. *(eq <expression> <expression>+)* retourne vrai si la première expression est égale à toutes les autres, *neq* vérifie que toutes les expressions sont différentes.
Les égalités entre valeurs numériques sont vérifiables avec les prédicats =<><=, etc.
D'autres prédicats plus complexes permettent, par exemple, de tester si l'ensemble des valeurs d'une expression est contenu dans une seconde expression (*(subsetp <expression-a-valeurs-multiples-1> <expression-a-valeurs-multiples-2>)*)
Le langage orienté objet de CLIPS dispose également de prédicats tel que *(instancep <expression>)* qui retourne vrai si l'expression est une instance de classe, ou encore *(superclassp <classe1> <classe2>)* qui retourne vrai si *classe1* est une superclasse de *classe2*. De plus, il est possible de développer ses propres prédicats. L'exemple suivant montre une règle qui vérifie qu'une valeur est numérique (prédicat numberp) et impaire (prédicat oddp)

```
(defrule test-nombre-impair
  (donnee ?x&:(numberp ?x)&:(oddp ?x))
  =>)
CLIPS> (assert (donnee 1) (donnee 2) (donnee rouge))
<Fact-2>
CLIPS> (agenda)
0      test-nombre-impair: f-0 ; la premiere valeur (donnee 1) est la
   seule valeur numerique impaire
For a total of 1 activation.
```

En se basant sur ces fonctions, on est capable de construire des tests et des suites de tests évolués. L'archive "features_test.zip" qui accompagne CLIPS en est une illustration. La suite de tests est définie dans un fichier "testall.tst" qui lance plusieurs tests unitaires définis dans des fichiers d'extension *.tst*. Un test :

- active la journalisation des opérations avec la commande *(dribble-on "nom-fichier-de-journalisation.out")*.
- lance un ensemble de commandes CLIPS contenues dans un fichier .bat avec la commande *(batch "nom-fichier.bat")*
- charge un fichier qui définit une fonction CLIPS qui compare le contenu de deux

fichiers passés en paramètre.

- invoque la fonction de comparaison pour comparer le contenu du fichier où les résultats d'exécution ont été enregistrés avec celui d'un fichier contenant les résultats attendus.
- la fonction de comparaison enregistre les éventuelles différences entre les deux fichiers dans un fichier .rsl (résultat).

Le principe est similaire à celui des tests unitaires JUnit en Java. CLIPS satisfait ainsi le critère de test des règles et obtient une note de 2 sur 2.

OpenRules

Le moteur de règles OpenRules peut être invoqué à partir de code Java, en spécifiant le fichier de tables décisionnelles principal à utiliser. La méthode *run* de la classe *OpenRulesEngine* permet d'exécuter le moteur de règles. Elle prend en paramètre la méthode à exécuter (définie dans des tables décisionnelles), ainsi qu'une liste d'objets sur lesquels les règles sont appliquées (données de test). La méthode à invoquer pourrait stocker les résultats d'exécution des règles dans une variable passée en paramètre. Le code Java appelant récupère alors ces résultats et vérifie qu'ils correspondent aux valeurs attendues.

La procédure au complet pourrait être définie dans un test unitaire, JUnit par exemple ; et répétée pour toutes les méthodes (règles) à tester, formant ainsi une suite de tests unitaires.

Le code suivant montre un test unitaire qui vérifie que, pour une cliente de sexe féminin mariée, la salutation est bien "Mme".

```
//Test qui verifie que la bonne salutation est retournee
@Test
public void salutationClient() {
  OpenRulesEngine engine = new OpenRulesEngine(nomDuFichierdeTables);
  //Donnee de test
  Client client = new Client();
  client.setNom("Zeineb");
  client.setSexe("Feminin");
  client.setEtatCivil("Marie");
```

```
    //Recuperer les resultats de l'execution dans un HashMap
    HashMap<String, Object> reponse = new HashMap<String, Object>();
    //Le moteur OpenRules prend une liste d'objets (donnees) comme
        parametres
    Object[] objets = new Object[] { client, reponse };
    //executer avec la methode a tester et les objects de test
    engine.run(nomMethodeATester,objets);
    //Verifier que les resultats sont corrects
    assertEquals("Mme", reponse.get("salutation"));
}
```

Étant donné qu'OpenRules permet de définir des types de données, des données et des méthodes directement dans des tables décisionnelles de type *Datatype*, *Data* et *Method* respectivement, on pourrait définir les tests dans des fichiers de tableur. En cachant les détails techniques, il serait alors aisé pour un utilisateur métier de saisir des données de tests et de vérifier la validité des résultats.
Les formulaires web, facilement définis avec OpenRules, constituent une interface alternative d'exécution de tests unitaires, accessible pour les utilisateurs métier.
Ainsi OpenRules peut utiliser les bibliothèques de test Java pour tester les règles d'affaires. Il peut aussi abstraire le code sous-jacent et présenter des interfaces de test accessibles aux utilisateurs métier. Le produit obtient une note de 2 sur 2 pour ce critère.

5.6.5 Journalisation et audit

Drools

drools-verifier[19] est un outil d'audit sophistiqué qui permet d'optimiser les règles, d'afficher la liste des faits manipulés par les règles ou encore de détecter les conditions non couvertes par les règles et les règles sans actions. Il détecte également les erreurs suivantes :

19. http ://community.jboss.org/wiki/DroolsVerifier

- redondances : deux règles avec les mêmes conditions et actions mais avec des attributs et des noms différents
- subsomptions : des règles avec les mêmes actions mais des conditions qui se chevauchent
- équivalences : deux règles équivalentes
- incohérences : la condition d'une règle est irréalisable (utilisation de contraintes contradictoires (a == b && a != b))
- toujours vraies : la condition de la règle est toujours vraie (a > b —— a <= b)

Les rapports générés par l'outil sont détaillés et des solutions concrètes sont proposées.

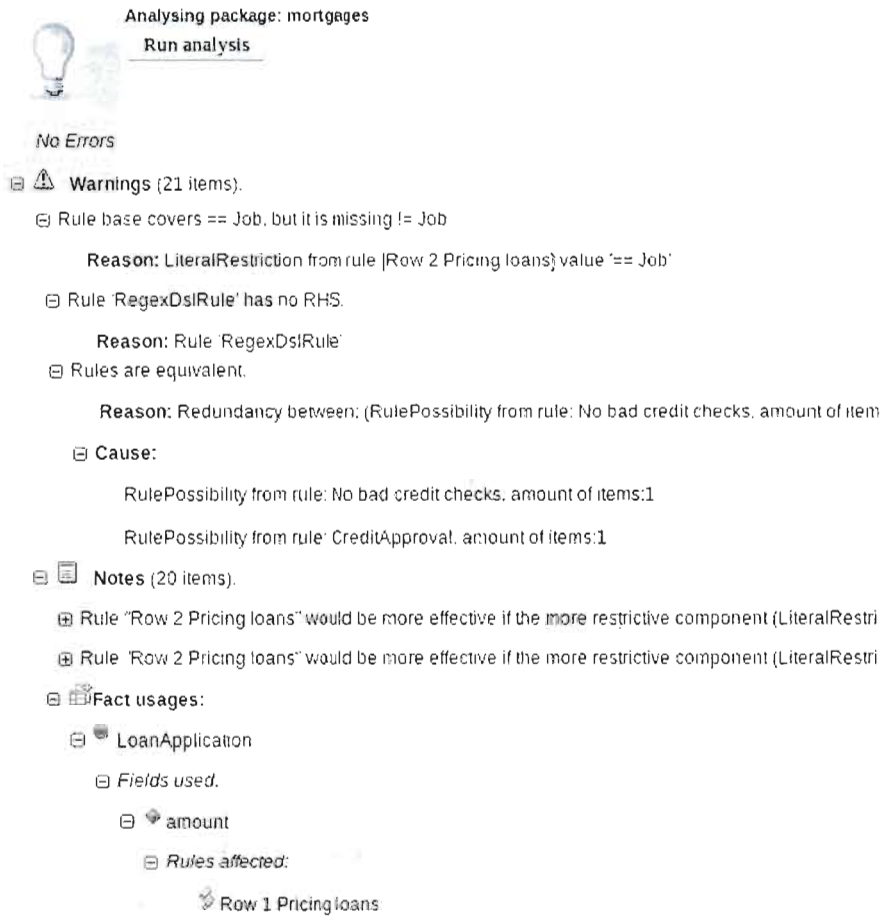


Figure 5.37 Analyse d'un paquetage de règles par Drools-Verifier

D'autre part, la vue d'audit affiche un journal d'audit créé à l'exécution du moteur de règles. Un journal d'audit est créé de la manière suivante :

```
WorkingMemory workingMemory = ruleBase.newWorkingMemory();
// Creer un nouveau logger pour la memoire de travail
WorkingMemoryFileLogger logger = new WorkingMemoryFileLogger(workingMemory
    );
// Un fichier evenements.log est cree dans le repertoire log
logger.setFileName( "log/evenements" );

workingMemory.assertObject(...);
workingMemory.fireAllRules();

// Arreter la journalisation
logger.writeToDisk();
```

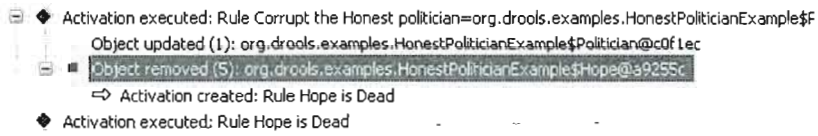


Figure 5.38 Vue *eclipse* du journal d'audit Drools

Un bouton *Open log* de la vue *eclipse* d'audit permet de sélectionner le fichier .log à consulter. La vue d'audit affiche les événements qui ont été journalisés dans le fichier sélectionné. Les événements sont représentés avec des icônes différentes selon leurs types (objet inséré, mis à jour ou retiré ; activation créée, annulée ou exécutée ; début ou fin d'un processus *Ruleflow* ; ajout ou suppression de règles...)
Chaque entrée contient des informations supplémentaires sur l'événement tel que son identifiant et sa représentation textuelle, les variables qui lui sont liées ou encore les noms des règles. Un événement qui survient pendant l'activation d'une règle est affiché comme un sous-nœud de l'activation de la règle.
Avec *drools-verifier*, la journalisation de l'exécution de règles et son affichage intuitif dans une vue *eclipse*, Drools répond pleinement à ce critère et obtient la note de 2.

CLIPS

CLIPS permet de générer des fichiers de journalisation avec la commande *(dribble-on <nom-fichier>)*. Toutes les informations habituellement envoyées à la console CLIPS, les messages d'avertissement, d'erreur et de suivi des entités de la base de connaissances sont enregistrés dans le fichier spécifié en paramètre. *(dribble-off* arrête l'envoi de données au fichier de journalisation.
Si les statistiques sont activées avec la commande *(watch statistics)*, plusieurs informations sont affichées après l'exécution de la commande *(run)* (temps d'exécution, nombre moyen de faits et d'activations...).

```
CLIPS> (watch statistics)
CLIPS> (run)
1 rules fired        Run time is 1.83689990080893e-05 seconds.
54439.547825095 rules per second.
3 mean number of facts (3 maximum).
2 mean number of instances (2 maximum).
1 mean number of activations (1 maximum).
```

En combinant les statistiques avec les commandes de débogage, on est capable de mesurer les temps d'exécution pour des règles données et déterminer ainsi les règles à optimiser et les éventuelles causes de ralentissement du système.
Ainsi, les commandes *(dribble-on)* et *(dribble-off)* fournissent une manière commode d'enregistrer toutes les actions survenues lors de l'exécution du moteur. L'affichage de statistiques combiné aux commandes de débogage fournit un outil d'audit puissant. Toutefois, le formatage des fichiers de journalisation est basique. CLIPS produit des fichiers de texte plats sans catégoriser les événements selon leur type, leur date ou encore le nom de l'utilisateur ayant entré les commandes. Les fonctionnalités de journalisation et d'audit sont réalisables avec CLIPS, mais restent perfectibles. Le produit obtient une note de 1 sur 2 pour ce critère.

OpenRules utilise la bibliothèque *log4j* pour assurer la journalisation. log4j permet une configuration avancée du niveau de détail des événements à journaliser (debug, info, warn, log...), du format d'affichage, ou encore du support à utiliser (sortie standard, fichier plat, fichier XML...). Un fichier de configuration *log4j.properties*, placé dans le même répertoire que les fichiers source Java du projet OpenRules, permet de définir les propriétés de la journalisation.

```
INITIALIZE OPENRULES ENGINE 4.1.2 for [file:/home/jihedamine/workspaces/java/HelloJava/rules/m
Registering data source handler {file : com.openrules.datasource.FileDataSourceFactoryHandler}
Registering data source handler {classpath : com.openrules.datasource.ClasspathDataSourceFacto
Set default data source handler {com.openrules.datasource.URLDataSourceFactoryHandler}
Create DataSource from file:/home/jihedamine/workspaces/java/HelloJava/rules/main/HelloCustome
[file:/home/jihedamine/workspaces/java/HelloJava/rules/main/HelloCustomer.xls] has been resolv
LANG: com.openrules.lang.xls
Registering data source handler {file : com.openrules.datasource.FileDataSourceFactoryHandler}
Registering data source handler {classpath : com.openrules.datasource.ClasspathDataSourceFacto
Set default data source handler {com.openrules.datasource.URLDataSourceFactoryHandler}
IMPORT.JAVA=hello.*
IMPORT.STATIC=com.openrules.tools.Methods
INCLUDE=../include/HelloRules.xls
```

Figure 5.39 Entrées dans un journal OpenRules

log4j est suffisamment puissant et paramétrable pour servir d'outil d'audit. Les événements peuvent être filtrés selon des critères avancés tel que leur date de déclenchement. Les filtres peuvent orienter le stockage des événements dans différents fichiers, afin de faciliter l'analyse des données.

Ainsi, OpenRules permet la journalisation via des bibliothèques tierces. Il ne fournit toutefois pas d'outils additionnels spécifiques aux systèmes de règles et c'est aux utilisateurs d'apprendre à maîtriser *log4j* pour réaliser une journalisation et un audit avancés des règles. Le produit obtient la note de 1 sur 2.

5.7 Le stockage des règles

5.7.1 Versionnage des règles

Drools

Guvnor dispose d'un dépôt de règles intégré. Le dépôt stocke exclusivement les règles d'affaires et les éléments qui leur sont liés (tables décisionnelles, modèles de données, définitions de DSLs, de fonctions...). Pour gérer les autres aspects de l'application, un système de contrôle de code source comme *subversion* ou *git* est recommandé.
Le dépôt Drools est basé sur *Apache Jackrabbit*. Par défaut, les données sont stockées dans un SGBD (Système de Gestion de Bases de Données) *Apache Derby* imbriqué. Il est cependant possible d'opter pour une base de données plus robuste tel que *Postgresql* ou *mySQL*.
Apache Jackrabbit est une implémentation de la spécification des dépôts de contenu pour Java (Content Repository for Java Technology API). Les dépôts de contenu sont des entrepôts de données permettant notamment la hiérarchisation, la recherche textuelle, le versionnage et les transactions.
Les dépôts Guvnor gèrent automatiquement le versionnage des règles. À chaque fois que des modifications sont enregistrées pour une règle donnée, son numéro de version est automatiquement incrémenté. Un volet *historique de version* affiche la version actuelle d'une règle ainsi que les versions précédentes et leurs dates de création. Il est possible de visionner la règle à une version précédente et de la restituer à cette ancienne version.
En plus du versionnage automatique, Guvnor définit la notion d'*état*. Chaque règle enregistrée dans le dépôt possède un état unique à un moment donné de son cycle de vie. Il est possible d'affecter un état à un paquetage entier de règles. On est alors capable d'archiver des règles ou des paquetages de règles à différents états et de les restituer ultérieurement à un état donné.
Drools offre plusieurs alternatives pour versionner les règles et restituer des versions précédentes. Il mérite une note de 2 pour ce critère.

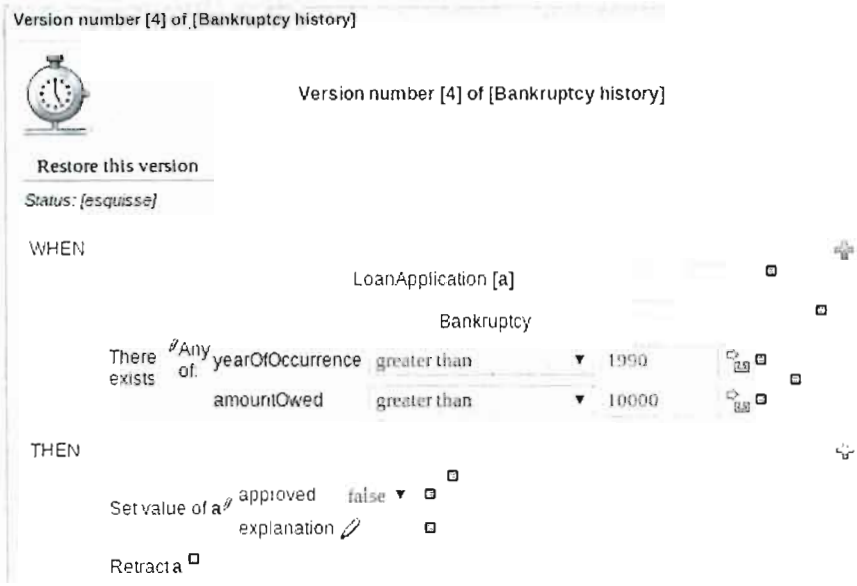


Figure 5.40 Visualisation d'une version précédente d'une règle

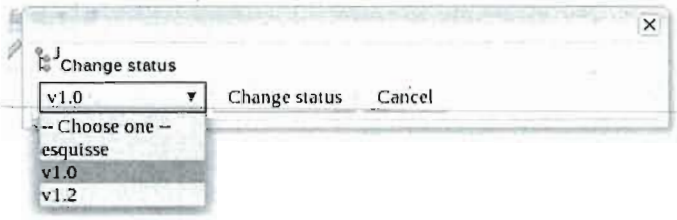


Figure 5.41 Changement de l'état d'une règle

CLIPS

CLIPS ne fournit pas de dépôt de données et ne prend pas en charge la notion de versionnage des règles. Il peut uniquement charger une suite de commandes CLIPS saisies dans des fichiers texte. Il obtient la note de 0 pour ce critère.

OpenRules

OpenRules peut accéder aux fichiers de type tableur stockés :

- sur le système de fichiers local (préfixes file et classpath)

- sur des serveurs d'applications distants (protocole http et ftp)
- dans des dépôts de contrôle de version (protocole http)
- dans une base de données relationnelle (préfixes db et dbv)

Le préfixe *dbv* est utilisé pour accéder à une base de données relationnelle avec prise en charge du versionnage. Le protocole *dbv* a été implémenté par OpenRules, en s'inspirant des systèmes de contrôle de version tel que *subversion.* Les fichiers de type tableur sont stockés dans une table *dbstorage*, dans un champ nommé *content* de type BLOB (objet binaire large). La clé primaire de la table est un champ *revision* de type entier, indiquant le numéro de version du fichier dans le dépôt. Les champs *name* et *timestamp* de la table indiquent respectivement le nom du fichier et sa date de création. Par exemple, *"dbv ://depotA :4/banque/Emprunts.xls"* permet d'accéder à la version numéro 4 du fichier *banque/Emprunts.xls*, stockée dans la base de données *depotA*. Si on ne spécifie pas de numéro de version, on récupère la version la plus récente du fichier. À chaque nouvel enregistrement d'un même fichier, le numéro de version est automatiquement incrémenté.

Le protocole dbv est défini dans le fichier *db.properties* d'un projet OpenRules :

```
openrules.protocol.name=dbv
openrules.protocol.class=com.openrules.dbv.DbvDataSourceHandler
db.implementation.class=com.openrules.dbv.admin.impl.RepoDAOImpl
```

La propriété *db.implementation.class* désigne une implémentation des commandes d'accès au dépôt :

- *db -r depotRegles -i* crée une base de données nommée *depotRegles* et contenant la table *dbstorage.*
- *db -ci /banque/Emprunts.xls -f /home/jihedamine/projetOpenRules/src/banque/Emprunt.xls -u admin -m "enregistrement initial"* permet de stocker une première version du fichier *Emprunts.xls* dans le dépôt en tant qu'*admin.*
- *db -l "*"* permet d'afficher le contenu du dépôt ainsi que le numéro de version courante.
- *Check-out* (ou -co) récupère un enregistrement, *Remove* (-rm) supprime un enregistrement, et *History* (-y) affiche toutes les versions du fichier qui lui est passé en

paramètre.

La figure 5.42 montre l'accès aux dépôts avec versionnage à partir de fichiers de type tableur, en spécifiant le fichier de configuration *db.properties* et l'URL d'accès au dépôt selon le protocole *dbv*.

Environment	
datasource	classpath:db.properties
include.path	dbv://depotRegles/
include	<banque/Emprunts.xls>
	<banque/Assurances.xls>
	<clients/Comptes.xls>

Figure 5.42 Accès à un dépôt avec versionnage à partir de tableurs

La documentation d'OpenRules réfère aux systèmes de contrôle de version tiers. On ignore cette possibilité dans notre évaluation car le produit ne fournit aucun interfaçage avec ces outils tiers. Un système tel que *subversion* ou *git* pourrait tout autant être utilisé pour versionner des fichiers plats utilisés par CLIPS.

Étant donné que le versionnage se fait à l'échelle des fichiers et non pas individuellement pour chaque table décisionnelle, on considère qu'OpenRules satisfait partiellement ce critère. Il obtient une note de 1 sur 2.

5.7.2 Hiérarchisation des règles

Drools

Guvnor affecte un attribut *catégories* aux règles et aux entités qu'il gère. Une catégorie est une sorte d'étiquette descriptive. La catégorie est différente de l'état car une entité passe par plusieurs états au cours de son cycle de vie, mais elle reste généralement dans la même catégorie (par exemple, la catégorie des règles d'hypothèque, d'emprunt..). De plus, une entité ne peut être que dans un seul état, mais peut appartenir à plusieurs catégories.

Chaque règle et chaque entité stockée dans le dépôt Guvnor doit appartenir au moins à une catégorie.

L'application web Guvnor permet d'ajouter, renommer et supprimer des catégories. Il est également possible de définir des sous-catégories.

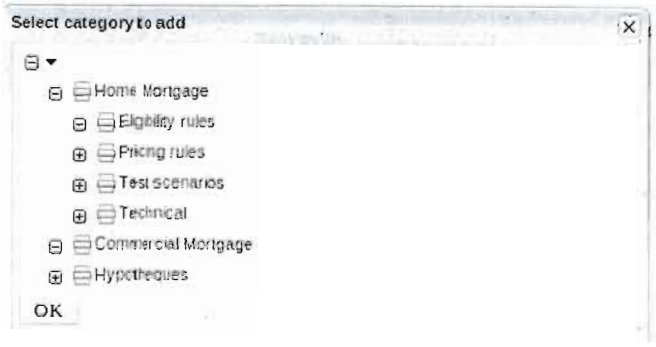


Figure 5.43 Catégories Guvnor

Les paquetages représentent une technique additionnelle d'organisation des règles et des entités. Leur fonctionnement est identique à celui des paquetages Java. Il est donc possible de créer, par exemple, un paquetage *banque.emprunt* sous-paquetage de *banque*. La notion de paquetages, contrairement à celle des catégories, n'est pas une extension de Guvnor mais elle est présente dans les définitions mêmes des règles (fichiers .drl, .dslr et tableurs).

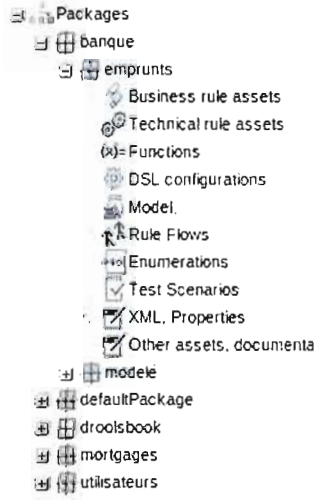


Figure 5.44 Hiérarchisation en paquetages Drools

Les catégories et les paquetages présentent deux moyens de hiérarchiser et de structurer les entités de la base de connaissances. Drools permet une administration avancée des deux aspects et obtient une note de 2 pour ce critère.

CLIPS

CLIPS permet de grouper les règles dans des modules. Cependant, on ne peut pas établir une hiérarchie de modules car il n'est pas possible d'imbriquer des sous-modules. Grâce au langage objet de CLIPS (COOL), on est capable d'établir une hiérarchie des faits en utilisant l'héritage de classes. On ne peut cependant pas créer une arborescence de règles. CLIPS obtient donc la note de 1 sur 2 pour ce critère.

OpenRules

OpenRules permet de définir une arborescence des fichiers de type tableur similaire à la notion de paquetages Java. L'arborescence est respectée, y compris lorsque les fichiers sont stockés dans une base de données relationnelle.
Comme pour les paquetages Java, il est possible d'imbriquer des sous-dossiers dans d'autres dossiers.

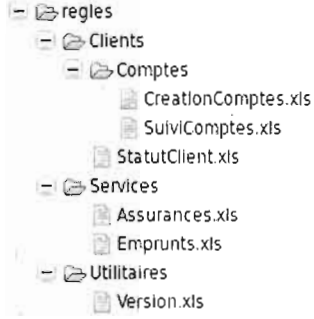


Figure 5.45 Hiérarchisation en dossiers OpenRules

Au sein d'un même fichier de type tableur, les tables peuvent être organisées dans des feuilles (*worksheet*) séparées par catégories. OpenRules satisfait donc le critère et obtient une note de 2 sur 2.

5.7.3 Définition de méta-données

Drools

Drools permet de déclarer et d'associer des méta-données aux faits en utilisant le symbole @ pour chaque nouvel identifiant de méta-donnée. Ces méta-données sont exploitées par les dépôts Guvnor.

```
declare Compte
  @auteur( jihedamine )
end
```

Le code précédent ajoute la méta-donnée *auteur* à l'entité *Compte*. L'entité peut être une classe Java ou un fait déclaré dans la base de connaissances Drools.
Par ailleurs, Guvnor définit des méta-données pour toutes les règles qu'il gère. Il s'agit d'informations sur les catégories auxquelles appartient une règle, la date de sa dernière modification, l'identifiant de la personne qui l'a modifié, la date de création et le format de la règle... On peut définir des méta-données additionnelles pour le sujet, le type et la source de la règle entre autres.
Toutes ces informations enrichissent la description des règles et peuvent être utilisées pour faire des recherches avancées (voir section suivante).
Drools propose donc des mécanismes de métadonnées extensibles pour la déclaration des faits. Par contre, les méta-données sur les règles ne peuvent pas être déclarées directement dans leurs définitions. De plus, l'interface proposée par Guvnor restreint le type de méta-données que l'on peut définir. Par conséquent Drools obtient une note de 1 sur 2 pour ce critère.

CLIPS

Les seules propriétés que l'on peut déclarer pour les règles CLIPS sont sa priorité (*salience*) et l'attribut *auto-focus*. Si une règle déclare *auto-focus* comme vrai, le module auquel elle appartient devient automatiquement le module courant lorsque la règle est

Figure 5.46 Méta-données d'une règle

activée dans l'agenda.

```
(defrule MODULEA::regle-1
   (declare (salience 10))
   (declare (auto-focus TRUE))
   (personne (nom ?nom) (age ?x&:(< ?x 0)))
   =>
   (printout t ?nom " a une valeur d'age non valide." crlf))
```

Hormis ces deux propriétés, CLIPS ne prend pas en charge la définition de méta-données pour les règles. On pourrait étendre le langage pour réaliser la fonction, mais tout devrait être fait à partir de zéro, dans un langage externe. Le produit obtient 0 pour ce critère.

OpenRules

La table *dbstorage* des dépôts avec versionnage d'OpenRules définit un champ *author* indiquant le nom de l'auteur du fichier de tables décisionnelles et un champ *comments* de type CLOB (longues chaînes de caractères) pour attribuer un commentaire au fichier.

Les logiciels de tableur (Excel, OpenOffice Calc..) permettent aussi de définir des commentaires par rapport à une cellule, à une ligne ou à toute la table. Contrairement aux champs *author* et *comments*, ces champs ne sont pas exploitables dans des requêtes de sélection.

Ainsi, OpenRules ne prend pas en charge le concept de méta-données mais permet de spécifier des champs textuels qui documentent les règles et les faits métier.

Le produit obtient une note de 1 sur 2, étant donné que les méta-données ne sont pas reconnues et exploitées comme telles par le système.

5.7.4 Recherche par critères

Drools

En plus du parcours intuitif des dépôts Guvnor par paquetages et par catégories, il est possible d'effectuer des recherches par mots-clés. Trois types de recherches sont possibles :

- recherche par nom d'entité : ce mode bénéficie de l'auto-complétion comme le montre la figure 5.47.
- recherche textuelle : le mot clé peut être le nom, la description ou toute autre information relative à l'entité recherchée.
- recherche multicritères : exploite les méta-données relatives aux règles pour permettre une recherche par auteur, par date de création ou encore selon la description de l'entité.

L'interface Guvnor ne permet pas de sauvegarder des requêtes pour une utilisation future. Cette fonctionnalité aurait permis de définir l'équivalent des vues utilisées dans les bases de données. Il est possible d'implémenter une telle fonctionnalité mais ceci nécessite un effort d'apprentissage des détails de l'API Drools et de développement d'interfaces destinées aux utilisateurs métier. Ainsi, le produit satisfait partiellement ce critère et obtient une note de 1 sur 2.

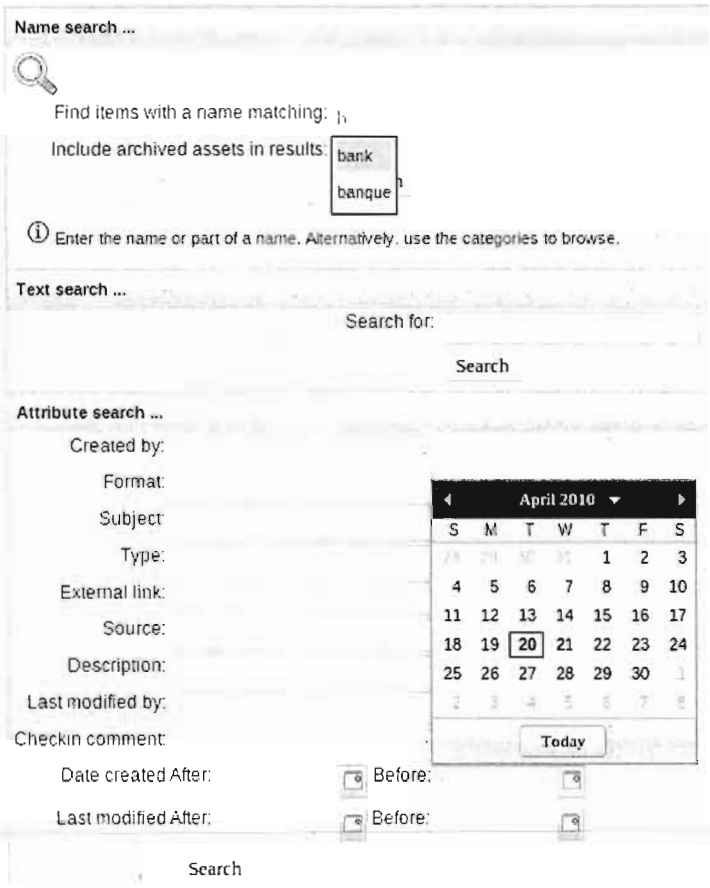


Figure 5.47 Recherche dans un dépôt

CLIPS

CLIPS permet d'effectuer des recherches sur l'ensemble de faits et d'instances de classes existants :

- any-factp/any-instancep : détermine si un ou plusieurs faits/instances satisfont à une requête.
- find-fact/find-instance : retourne le premier fait/instance qui satisfait la requête.
- find-all-facts/find-all-instances : groupe et retourne tous les faits/instances qui satisfont une requête.
- do-for-fact/do-for-instance : effectue une action pour le premier fait/instance qui sa-

tisfait une requête.

- do-for-all-facts/do-for-all-instances : effectue une action pour tout fait/instance qui satisfait une requête.
- delayed-do-for-all-facts/delayed-do-for-all-instances : groupe tous les faits/instances qui satisfont une requête puis applique itérativement une action sur le groupe.

Les deux exemples suivants illustrent l'utilisation des commandes de recherche. Le premier exemple retourne vrai s'il existe un fait de type homme d'âge supérieur à 30. Le deuxième exemple retourne toutes les paires d'hommes et de femmes qui ont le même âge.

```
CLIPS> (any-factp ((?homme homme)) (> ?homme:age 30))
TRUE

CLIPS>(find-all-instances ((?h HOMME) (?f FEMME)) (= ?h:age ?f:age))
([Homme-1] [Femme-1] [Homme-2] [Femme-2])
```

S'il est possible de rechercher des faits et des instances de classes avec des commandes CLIPS, on n'est pas capable d'en faire de même pour les règles. L'impossibilité de définir des méta-données réduit sensiblement les possibilités de recherche. Ainsi, CLIPS n'obtient que 1 sur 2 pour ce critère.

OpenRules

Les dépôts de bases de données avec versionnage offrent la commande List qui prend en paramètre un masque de nom de fichier pouvant inclure des caractères "joker" tel que '*' et '?'. Un paramètre optionnel *-rev* permet de spécifier le numéro de version maximal du fichier à récupérer.

Par exemple *db -l "banque/*" -rev 9* permet de récupérer tous les fichiers dont le nom complet commence par "banque/" et dont le numéro de version est inférieur ou égal à 9.

Il est également possible d'exploiter les champs *author*, *comments* et *timestamp* pour effectuer des requêtes de sélection par nom d'auteur, date de création ou commentaires

associés au fichier.

Pour ce qui est des bases de données non versionnées, les requêtes de sélection SQL sont également utilisables. De même, il est possible de définir des vues pour enregistrer des requêtes de sélection et les ré-exécuter ultérieurement.

En définissant une table décisionnelle par fichier de type tableur, on obtient une granularité de recherche plus fine, mais on pénalise les performances du système.

De plus, on pourrait définir des champs additionnels dans les dépôts de bases de données pour stocker des méta-données supplémentaires et permettre ainsi une recherche plus avancée, avec plus de critères à disposition.

Ces options sont simples à implémenter mais ne sont pas fournies par OpenRules. La granularité par défaut est élevée puisque les sélections se font au niveau des fichiers et non au niveau des tables décisionnelles. Ainsi, le produit satisfait partiellement le critère et obtient la note de 1 sur 2.

5.7.5 Administration des dépôts

Drools

Guvnor permet une administration avancée des dépôts.

Les utilisateurs ayant des permissions d'administrateur Guvnor ont accès à un volet *administration* qui permet :

- la gestion des catégories et des états.
- l'archivage et la restauration d'entités archivées.
- la consultation d'un journal des manipulations du dépôt : chaque entrée indique le nom de l'utilisateur, l'opération qu'il a effectuée et sa date.
- la gestion des permissions d'accès au dépôt (voir le critère *5.4.3 Interfaces pour non informaticiens*).
- l'importation et l'exportation de dépôts : quelque soit le système de stockage en arrière plan, les dépôts Guvnor offrent une couche d'abstraction permettant l'importation et l'exportation de dépôts sous format XML.

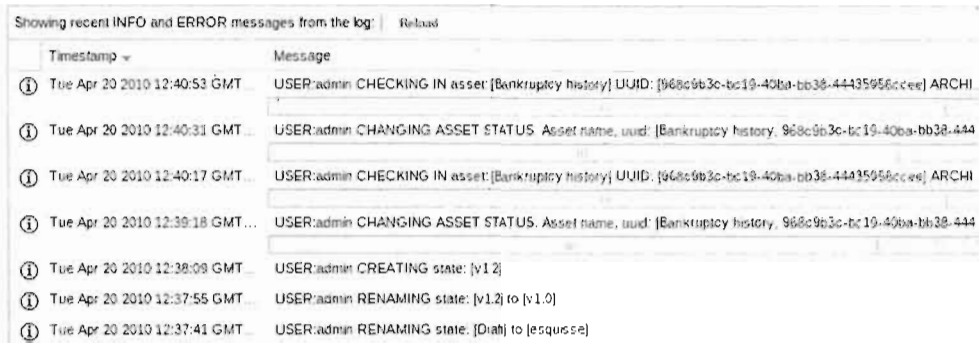


Figure 5.48 Journal des actions des utilisateurs du dépôt Guvnor

Par ailleurs, le contenu des dépôts est accessible via l'interface web de Guvnor, via WebDAV ou encore avec les flux *atom*. WebDAV est un ensemble d'extensions du protocole de transport HTTP qui permet de manipuler et de gérer de manière collaborative des fichiers stockés sur des serveurs web distants.
Le plugiciel *eclipse* permet la synchronisation de fichiers entre les dépôts Guvnor, l'espace de travail *eclipse* et d'éventuels systèmes de contrôle de code source.
La figure 5.49 montre la vue *eclipse Guvnor Repositories* qui permet de parcourir un dépôt Guvnor via WebDAV. S'il existe un éditeur *eclipse* pour un type de fichier présent dans le dépôt, il est automatiquement utilisé pour la visualisation en lecture seule.
Guvnor permet une administration avancée des dépôts et obtient une note de 2 sur 2.

CLIPS

CLIPS fournit un moteur de règles et non pas un système de gestion de règles, il ne propose donc pas de dépôt de règles. Les règles peuvent être écrites dans des fichiers texte, mais on ne peut parler "d'administration de dépôt". CLIPS obtient 0 pour ce critère.

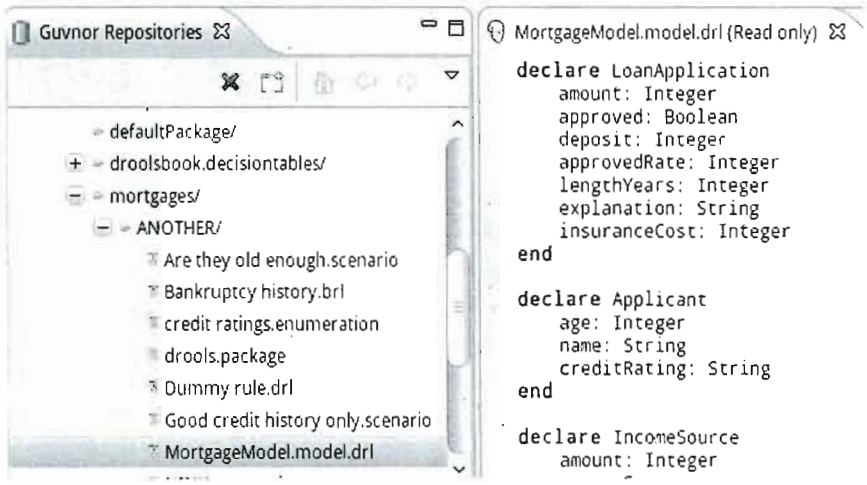


Figure 5.49 Accès WebDAV à un dépôt Guvnor à partir d'*eclipse*

OpenRules

En utilisant les bases de données relationnelles, OpenRules permet une administration avancée des dépôts : Il est possible de synchroniser, importer, exporter, répliquer, sauvegarder et restaurer des dépôts de fichiers de tables décisionnelles.

De plus, on peut implémenter des dépôts OpenRules sur plusieurs types de supports physiques : système de fichiers local, serveur d'applications distant, système de contrôle de version et système de gestion de bases de données relationnelles.

On note que les options d'administration disponibles pour les bases de données ne le sont pas pour tous les autres types de supports. Contrairement à Drools, OpenRules ne gère pas les permissions d'accès aux dépôts et ne propose pas une interface graphique intuitive pour l'accès aux dépôts et leur administration.

Ainsi, l'administration des dépôts n'est possible que pour certains supports physiques. En dehors des commandes d'accès aux dépôts de bases de données avec versionnage, OpenRule ne fournit pas d'outils d'administration mais réfère à des outils tiers.

La note attribuée sur ce critère est de 1 sur 2.

5.8 Performance et coût

5.8.1 Performance des moteurs de règles

Drools

Illation est une compagnie experte dans l'évaluation des performances des moteurs de règles. En plus des bancs d'essais discutés à la section *3.2.1 Performance et coût en ressources*, elle a développé un banc d'essai bancaire qui fait la somme des flux de trésorerie de crédit et de débit pour des périodes comptables et détermine le solde d'un compte à la fin de chaque période. Contrairement au banc Miss Manners où plusieurs activations de règles sont mises dans l'agenda et une seule est sélectionnée pour exécution, ce banc d'essai force le moteur à exécuter toutes les activations.
Illation a publié [20] les résultats de plusieurs bancs d'essais, comparant les performances de :

- OPSJ 6, Drools 4 et Drools 5 selon le banc d'essai Waltz : Drools 4 est plus rapide, consomme moins de mémoire et charge les données plus rapidement que Drools 5. OPSJ est meilleur sur tous les plans que les deux versions de Drools.
- OPSJ 6, Drools 4 et Drools 5 selon le banc d'essai Waltz DB : Drools 5 est nettement plus rapide que Drools 4 mais consomme plus de mémoire. OPSJ est plus rapide que les deux versions de Drools et consomme moins de mémoire.
- OPSJ 6, Drools 4 et Drools 5 selon le banc d'essai Miss Manners : Drools 5 est plus rapide que Drools 4 et les consommations mémoire sont égales et moins importantes que pour OPSJ 6. Cependant, les deux versions de Drools ont été incapables de résoudre le problème pour un nombre de centres d'intérêt supérieur à 3.
- Jess 7, OPSJ 6, Drools 4 et Drools 5 selon le banc d'essai bancaire : Globalement, Drools 5 a été plus performant que Drools 4. OPSJ6 est clairement gagnant sur tous les aspects et Jess 7 arrive dernier sauf pour la consommation mémoire.

---

20. http ://blogs.illation.com.au

Globalement, les performances de la version 5 de Drools sont meilleures que celles de la version 4 et OPSJ surpasse les deux versions quasiment partout.
Les bancs d'essais ne donnent pas une idée globale sur les performances des moteurs de règles dans un environnement de production. Les variations dans les résultats d'un banc d'essai à un autre en témoignent.
Ce qui ressort des retours d'information des utilisateurs de Drools est que le moteur présente des performances honorables, sans plus. De l'aveux des développeurs eux-mêmes, les performances sont "suffisamment bonnes". Ils notent qu'une mauvaise approche des utilisateurs est souvent la cause de la lenteur des temps d'exécution mais concèdent que beaucoup d'améliorations peuvent être apportées aux performances intrinsèques du moteur [21].
Il est envisagé pour Drools 6 d'implémenter un appariement de modèles orienté collections, améliorant ainsi l'évaluation parallèle de règles [22].
En se basant sur les résultats des bancs d'essais, sur les retours d'information des utilisateurs et sur les commentaires des développeurs, on estime que Drools mérite une note de 1 sur 2 pour les performances du moteur de règles.

CLIPS

La compagnie Illation a publié des tests de performances pour les deux versions les plus récentes de CLIPS (6.24 et 6.30), comparées à celles des moteurs de règles commerciaux Blaze Advisor 6.5, ILog JRules, JESS 7.0p1 et JBoss Rules 4.0 (version de Drools avec assistance professionnelle) [23].

- Le banc d'essais Manners a été effectué sur trois plates-formes différentes (Linux, Solaris et Windows). L'ancienne version de CLIPS (6.24) a eu de mauvais résultats. Par contre, CLIPS 6.30 a produit la meilleure performance pour les ensembles de

21. http ://blog.athico.com/2008/09/rule-engines-and-performance.html

22. http ://blog.athico.com/2010/02/collection-oriented-match-for-massively.html

23. http ://illation.com.au/benchmarks/ruletesting.html

données de 16 et 32 invités, suivi de près par Blaze Advisor et dépassant largement les autres produits. Pour les variantes de Manners avec 64 et 128 invités, CLIPS 6.30 arrive juste derrière Blaze Advidor et loin devant les autres produits.

- Le banc d'essais Waltz a été exécuté avec des ensembles de données de 12, 25, 37 et 50 régions. CLIPS 6.24 a eu de mauvais résultats. Par contre, CLIPS 6.30 arrive encore une fois en pole position pour toutes les variantes du banc d'essais sauf celle à 50 régions.
- Le banc d'essais WaltzDB a été exécuté avec des ensembles de données de 4, 8, 12 et 16 régions. Ici aussi, CLIPS 6.30 est plus rapide que tous ses concurrents commerciaux ; et ce pour toutes les variantes du banc d'essais.

Par ailleurs, un nouveau banc d'essai, nommé Sudoku, demande aux moteurs de règles de résoudre des grilles de Sudoku de 3x3, 4x4 et 5x5 cases et avec des niveaux de complexité différents. Les tests ont été effectués avec les moteurs de règles CLIPS 6.24, CLIPS 6.30 et JESS 70p1. CLIPS 6.30 a obtenu de meilleurs résultats pour tous les tests, suivi de JESS 70p1. CLIPS 6.24 a été le moins performant. CLIPS 6.30 a résolu deux grilles de 4x4 et 3x3 de même complexité en respectivement 4.54 et 0.58 secondes alors que JESS 70p1 en a mis 30,6 et 8.80. La différence est donc notable.

Ce qui ressort de tous ces bancs d'essais, c'est que CLIPS 6.30 surpasse souvent les moteurs de règles commerciaux les plus performants. Cette dernière version a bénéficié de larges améliorations de performances par rapport à la version précédente, plus lente que ces autres concurrents.

Toutes les structures CLIPS (règles, modèles de faits, faits...), une fois chargées à partir d'un fichier, sont stockées en mémoire. Si elles sont chargées puis enregistrées sous format binaire avec la commande *bsave*, il est alors possible de les lancer du fichier binaire avec la commande *bload*, avec des temps de chargement beaucoup plus courts qu'avec la commande *load*. Le format binaire permet de charger simultanément des blocs entiers de constructions au lieu d'avoir à lire et interpréter individuellement des chaînes de caractères dans un fichier texte.

L'homogénéité des résultats obtenus par CLIPS 6.30 pour les divers bancs d'essais, ainsi que le large écart qui le sépare de la quasi-totalité de produits commerciaux concurrents

et la possibilité de charger les constructions sous format binaire nous permettent d'affirmer que le moteur satisfait pleinement les critères de performance. Il obtient la note de 2 sur 2 pour ce critère.

OpenRules

OpenRules ne dispose pas d'un moteur d'inférence et n'utilise pas d'algorithme de chaînage pour exécuter les règles. Ainsi, les concepts d'agenda d'exécution, d'activation de règles et d'appariements partiels n'existent pas. Le système exécute une suite d'instructions Java selon une approche impérative "classique". Par suite, les temps d'exécution reposent sur les performances de la machine virtuelle Java et non sur un moteur d'inférence dédié.

L'algorithme d'exécution des règles est très naïf. Pour les tables décisionnelles sans valeur de retour, tous les enregistrements sont vérifiés, les uns à la suite des autres. Ceci pénalise fortement les temps d'exécution pour les tables volumineuses où seulement quelques règles sont satisfaites.

De plus, le moulinage du contenu des fichiers de type tableur requiert un temps de traitement additionnel.

Enfin, l'utilisation d'expressions Java dans les tables OpenRules détériore les performances d'exécution. En effet, pour chaque cellule comportant une expression, OpenRules crée de nouvelles instances des classes Java correspondantes [24].

Le site d'OpenRules indique que le système affiche de bonnes performances en matière de parallélisation et de mise à l'échelle. Le site se base sur un rapport effectué par une banque en 2006 pour évaluer les performances d'OpenRules. Il est indiqué que la banque a finalement sélectionné OpenRules et que les résultats des tests sont disponibles sur demande [25].

On estime qu'un cas particulier, publié par les auteurs du produit, ne peut constituer une

24. http ://openrules.com/docs/NL.htm#Performance%20Considerations

25. http ://openrules.com/overview.htm#Performance%20and%20Scalability

preuve suffisante du niveau de performance du système. Les bancs d'essais généralement utilisés pour évaluer les performances des moteurs de règles ne sont simplement pas applicables pour OpenRules, qui ne dispose pas d'un moteur d'inférence.
Une note de 1 sur 2 est attribuée au produit.

5.8.2 Coûts en ressources

Drools

Les bancs d'essais Drools cités précédemment évaluent la consommation mémoire en plus des temps d'exécution. Il s'en dégage que la version 5 de Drools consomme plus de mémoire que la version 4. Comparé à d'autres moteurs de règles, Drools n'est pas celui qui consomme le moins de mémoire.
Le modèle communautaire et à code ouvert de Drools a l'avantage de rapidement faire remonter ce genre d'informations aux développeurs. Des utilisateurs ayant un large volume de données se sont plaints d'une utilisation mémoire conséquente et d'un mécanisme de ramasse-miettes qui n'arrive pas à suivre la vitesse d'allocation des objets. En conséquence, les développeurs du produit ont récemment mis à jour leur algorithme Rete OO.
Pour faire une mise à jour, l'ancien algorithme parcourait le réseau Rete à deux reprises, une fois pour rétracter un fait et une autre fois pour en insérer un nouveau. Le réseau d'appariements partiels formant l'arbre Rete était donc détruit et reconstruit à chaque fois. Dans le cas où seule une petite portion de l'arbre était mise à jour, il s'agissait clairement d'un gaspillage de ressources. Le nouvel algorithme, nommé *True modify*, parcoure une seule fois le réseau et tente de préserver et de réutiliser les appariements partiels encore vrais après la mise à jour.
Cette modification de l'algorithme est sensée améliorer les performances et réduire la consommation mémoire. Toutefois, elle n'est pas encore déployée dans la version stable de Drools et l'amplitude des améliorations reste à évaluer.
Pour ce qui est des besoins en espace disque, la distribution binaire de Drools déployée

occupe un espace disque de 60.7 MB, ce qui est raisonnable.

Les plugiciels *eclipse* nécessitent l'installation d'une version récente de l'IDE (<= *eclipse* 3.4). Une machine avec un processeur Pentium récent et un minimum de 256 MB de RAM sont requis pour exécuter l'IDE.

Par ailleurs, Drools Guvnor doit être déployé sur un serveur d'applications Java EE (préférablement JBoss Application Server), ce qui engendre des coûts additionnels en ressources.

Ainsi, Drools affiche des résultats moyens concernant les coûts en ressources et n'obtient que 1 sur 2 pour ce critère.

CLIPS

L'exécutable CLIPS installé sur un système Linux n'occupe que 670 KB d'espace sur le disque dur. Les archives des fichiers exécutables pour Windows et Mac disponibles sur le site du projet ne font que 883 KB et 544 KB respectivement. Les besoins en espace de stockage sont donc minimes.

Concernant la consommation de mémoire, CLIPS fournit des commandes pour mesurer le nombre d'octets actuellement utilisés ou réservés par le moteur de règles avec la commande *(mem-used)*. *(mem-requests)* retourne le nombre de requêtes mémoire envoyées par CLIPS au système d'exploitation. Si le coût d'une allocation mémoire du système est connu, il est alors possible de calculer la taille totale de mémoire utilisée par CLIPS avec la formule : (+ (mem-used) (* <espace-alloue-en-octets> (mem-requests))).

Par ailleurs, *(release-mem)* libère toute la mémoire réservée et non utilisée par CLIPS. Lorsqu'il manque de mémoire, le moteur de règles invoque automatiquement cette commande, permettant au système d'exploitation de regrouper de petits blocs mémoire en des blocs plus grands. La commande retourne le volume de mémoire rendu par CLIPS au système d'exploitation.

Enfin, *(conserve-mem on/off)* active/désactive le stockage d'informations pour les commandes d'affichage et d'enregistrement. Dans le cadre de systèmes volumineux, ceci permet d'économiser un volume considérable de mémoire.

Sur les forums du produit, les utilisateurs rapportent une faible consommation mémoire dans des environnements de production importants. Par exemple, un intervenant affirme exécuter le moteur avec 30000 faits et des règles indépendantes sans aucun problème de performances sur une machine virtuelle VMWare disposant de 512 MB de mémoire vive.

Le faible coût du stockage, les multiples commandes de gestion avancée de la mémoire ainsi que les retours d'information positifs des utilisateurs de CLIPS font qu'on estime que le produit satisfait pleinement ce critère. Une note de 2 sur 2 lui est accordée.

OpenRules

L'installation d'OpenRules regroupe les bibliothèques du produit dans quatre dossiers :

- openrules.config : libraires OpenRules dont les bibliothèques tierces tel que Apache POI pour le moulinage des tables décisionnelles.
- openrules.forms.lib : bibliothèques pour générer les formulaires web .
- lib.jsr94 et com.openrules.jsr94 : bibliothèques JSR-94 et leur implémentation OpenRules.

L'ensemble occupe un espace de 7 MB sur le disque dur. Si on rajoute les exemples fournis avec le produit, on atteint 18 MB, ce qui est raisonnable.

La définition des tables décisionnelles requiert l'installation d'un tableur (Excel ou OpenOffice Calc) qui occupent de 20 à une centaine de MB d'espace disque. L'alternative Google Spreadsheet ne requiert aucune installation mais une connexion internet est nécessaire.

Par ailleurs, les plugiciels *eclipse* nécessitent l'installation d'une version récente de l'IDE (<= *eclipse* 3.2). Une machine avec un processeur Pentium récent et un minimum de 256 MB de RAM sont requis pour exécuter l'IDE.

On estime que l'exécution de l'IDE *eclipse* ou d'une application de tableur engendrent une consommation mémoire moyenne, qu'un poste de travail récent est capable de supporter.

Les exigences en espace de stockage et en consommation mémoire étant supérieures à

celles de CLIPS et équivalentes à celles de Drools, le produit obtient la note 1.

5.9 Prise en main

5.9.1 Documentation

Drools

Drools offre une documentation officielle librement accessible sur internet[26]. En plus d'un document d'introduction générale, le moteur de règles, les modules Guvnor, Flow (flux de règles), Fusion (traitement d'événements complexes) et Solver (problèmes de planification) possèdent chacun leur propre documentation. Le style y est très pédagogique et l'approche est graduelle, prenant en main le lecteur des concepts fondamentaux vers des aspects plus complexes. Le tout est étayé d'illustrations et d'exemples simples et expressifs.

En plus de la documentation officielle, deux livres se portent sur la dernière version de Drools. Le guide "Drools JBoss Rules 5.0 Developer's Guide" (Bali, 2009) est destiné aux développeurs et comprend beaucoup d'exemples illustrant les fonctionnalités Drools. On a cependant trouvé certaines erreurs dans le code et un manque de profondeur des explications. Le second livre (Browne, 2009) est destiné aux utilisateurs métiers et traite principalement de l'application Guvnor et des plugiciels *eclipse*. Un effort est fait pour abstraire les notions techniques et trouver des analogies parlantes pour les utilisateurs métier. Le livre explique certaines notions techniques de base nécessaires à la manipulation du produit. Ainsi Drools bénéficie d'une documentation exhaustive et généralement de bonne qualité, destinée aussi bien aux informaticiens qu'aux utilisateurs métier. Une note de 2 sur 2 est accordée pour ce critère.

---

26. http ://www.jboss.org/drools/documentation.html

CLIPS

Le manuel utilisateur de CLIPS (CLIPS User's Guide 2.0) est conçu comme une introduction au langage de programmation CLIPS claire et facile à lire. Il contient plusieurs exemples courts qui peuvent être vérifiés en les saisissant depuis la ligne de commande du système. Le manuel est complet mais sa lecture prend du temps. Il constitue un bon moyen d'apprendre la syntaxe du langage. Le manuel de référence CLIPS (CLIPS Basic 6.24) traite de sujets plus avancés qui ne sont pas explicitement mentionnés dans le manuel utilisateur. Les deux documents sont gratuitement téléchargeables sous format PDF. Le manuel utilisateur comporte 154 pages et le manuel de référence 398 pages. Il existe également un guide d'utilisation des interfaces graphiques (114 pages), un manuel d'architecture (457 pages) et un guide de programmation avancée qui documente exhaustivement l'API C d'accès à CLIPS (205 pages).
La documentation est très complète et existe pour tous les niveaux techniques. À l'intérieur d'un même manuel, l'information est bien structurée et étayée de nombreux exemples. Le produit obtient une note de 2 sur 2 sur ce critère.

OpenRules

La documentation d'OpenRules est disponible gratuitement sur internet[27]. Elle comprend un guide d'installation détaillé et un document principal bien structuré, comprenant une introduction, un exemple illustratif et des sections sur les tables décisionnelles, les formulaires web, les dépôts et l'API OpenRules, entre autres.
Toutes les sections bénéficient de figures et d'exemples illustratifs clairs. Les exemples sont variés et portent sur tous les aspects du produit (modèles de règles, dépôts, données XML...). Un scénario type simule l'interaction informaticien-expert métier pour l'élaboration d'un système de règles dans le domaine médical. Le texte est principalement destiné aux experts métier et prend le temps d'expliquer le jargon technique.
OpenRules fournit donc une documentation de qualité et obtient la note de 2 sur 2.

---

27. http ://openrules.com/docs/man.html

5.9.2 Convivialité

Drools

L'installation de Drools se fait simplement en désarchivant un fichier .zip qui contient l'ensemble des bibliothèques Java du produit. L'installation de Guvnor peut se faire en déployant une archive *drools-guvnor.war* dans un serveur d'applications selon une procédure décrite dans la documentation officielle. Cette procédure peut échouer en raison de bibliothèques manquantes au serveur d'applications ou ayant des versions incompatibles avec les besoins de Guvnor. Par ailleurs, le déploiement d'une application web demande des connaissances techniques dont ne disposent généralement pas les utilisateurs métier. Il existe à cet effet un paquetage comprenant un serveur d'applications JBoss Application Server dans lequel Guvnor est déjà déployé. Le paquetage est téléchargeable sur le site de Drools et il est prêt à l'emploi sans aucun besoin de configuration.

Les plugiciels *eclipse* de Drools sont facilement installables. Il suffit d'indiquer dans *eclipse* l'adresse URL de leur site d'installation et de choisir les modules que l'on désire installer.

Un utilisateur métier devrait donc être capable d'installer tous les composants du produit sans grandes difficultés. Pour ce qui est de la convivialité des interfaces, Guvnor y porte une attention particulière. Des icônes illustrent les différents éléments, les indications sont claires et les opérations bien hiérarchisées.

Les plugiciels *eclipse* sont également intuitifs. Les éditeurs graphiques de règles et de flux sont bien conçus et permettent d'être productif.

Il est clair que Drools a soigné la procédure d'installation et les interfaces de son produit afin de répondre aux besoins des utilisateurs métiers. La note obtenue pour ce critère est donc de 2 sur 2.

CLIPS

L'installation de CLIPS se fait avec un installateur simple d'accès, disponible pour les systèmes d'exploitation majeurs (Windows, Mac, Linux...). Des raccourcis sont automatiquement créés pour accéder directement à la ligne de commande et aux interfaces graphiques. Ces dernières sont intuitives et disposent d'assistants, mais elles restent assez basiques.

Même si l'accès au système est aisé, il s'agit d'un moteur de règles et non d'un système de gestion de règles. Les interactions se font essentiellement avec des commandes du langage, ce qui pourrait rebuter des non informaticiens. L'absence de dépôts de règles et d'interfaces graphiques évoluées similaires à Guvnor abaissent la convivialité du produit, qui obtient une note de 1 sur 2.

OpenRules

L'installation d'OpenRules est illustrée par un guide d'installation détaillé pour non informaticiens.

Deux types d'installation existent :

- utilisation du plugiciel *eclipse* : un site de mise a jour permet d'installer le plugiciel. Ce dernier télécharge automatiquement les bibliothèques OpenRules et ajoute les nouveaux types de projets à l'environnement *eclipse*.
- téléchargement d'une archive contenant les bibliothèques OpenRules et son extraction sur le disque dur.

Les exemples fournis avec OpenRules ainsi que les nouveaux projets créés par l'utilisateur sont accompagnés de fichiers .bat (run.bat, compile.bat...) pour lancer l'exécution dans l'environnement Windows. Cette fonctionnalité est destinée aux utilisateurs métier et les dispense d'apprendre à invoquer des cibles ant.

De plus, des fichiers *readme* détaillés accompagnent les exemples et expliquent leur structure et leur fonctionnement aux non informaticiens.

Par ailleurs, les applications de tableurs sont intuitives à utiliser et sont largement

adoptées en entreprise.
Enfin, le plugiciel *eclipse* s'intègre bien à l'IDE et permet de rapidement créer et exécuter des projets OpenRules.
L'installation du produit est simple et ses interfaces sont intuitives. Il obtient une note de 2 sur 2.

5.9.3 Courbe d'apprentissage

Drools

Drools présente des concepts nouveaux tel que les fichiers de règles .drl, les flux de règles et les dépôts Guvnor. La maîtrise du produit nécessite donc un certain temps d'apprentissage. De plus, les fonctionnalités avancées tel que la définition d'actions complexes dans les règles et la création de tests JUnit demandent une bonne connaissance du langage Java.
Pour faciliter son apprentissage, Drools offre une documentation exhaustive et pédagogique comprenant plusieurs exemples. Par ailleurs, des éditeurs graphiques et des assistants permettent la définition des règles et des flux. Guvnor offre plusieurs métaphores pour abstraire les notions Java tel que les types de variables, les tests unitaires, etc.
La courbe d'apprentissage de Drools est donc relativement faible et un utilisateur non expérimenté est capable de maîtriser le produit dans un temps raisonnable. La note accordée est de 2 sur 2.

CLIPS

CLIPS dispose d'un langage spécialisé et d'une approche qu'on estime intuitive. Des concepts clairs tel que les faits, les modèles de faits, les règles, etc, sont pris en charge directement par le produit et permettent de construire des systèmes experts complexes assez rapidement. La documentation est également pédagogique et exhaustive, en plus d'être disponible gratuitement.
Si la courbe d'apprentissage du fonctionnement de base du système reste faible, la

réalisation d'opérations évoluées tel que l'intégration à un système existant ou l'extension avec un langage de programmation externe nécessitent un effort d'apprentissage considérable. L'API d'interfaçage doit être maîtrisée et plusieurs obstacles risquent de survenir dans la manipulation des engrenages de bas niveau du moteur.
Ces cas d'utilisation étant avancés, on estime que la courbe d'apprentissage pour une utilisation typique de CLIPS reste globalement faible. Le produit obtient la note de 2 sur 2.

OpenRules

OpenRules préconise d'attribuer la définition des règles et des entités métier aux experts métier (tables décisionnelles) et l'implémentation des détails techniques aux informaticiens (code Java). Si l'approche est adoptée par l'organisme, la courbe d'apprentissage pour les experts métier est quasi-nulle, étant donné qu'ils sont habitués aux tables décisionnelles. La séparation de la logique métier du code applicatif permet de dispenser les experts métier d'apprendre à écrire des instructions Java ou de se soucier de l'exécution du moteur OpenRules.
En ce qui concerne les informaticiens, ils se retrouvent dans un environnement familier. OpenRules ne définit pas un langage de programmation nouveau et utilise des bibliothèques Java connues et largement utilisées (Apache POI, log4j...). Ainsi, un développeur connaissant Java pourra être opérationnel avec OpenRules en quelques heures.
On note toutefois que la collaboration entre les deux types de connaissances est nécessaire. Sinon, l'utilisateur métier devra assimiler des notions de programmation, ce qui rallongera le délai d'apprentissage du produit.
Globalement, la courbe d'apprentissage d'OpenRules est faible. Il obtient une note de 2 sur 2.

5.10 Évaluation sur l'aspect logiciel libre

5.10.1 Communauté

Drools

Drools est un projet communautaire ouvert aux suggestions et aux contributions de tous ses utilisateurs. Le blog officiel[28] est constamment mis à jour et les commentaires des utilisateurs sont nombreux. Des conférences sont régulièrement organisées pour réunir la communauté et présenter les nouveautés. Le produit dispose d'une liste de courriel (mailing list) où s'échangent en moyenne une dizaine de messages par jour, ce qui constitue un volume considérable pour un produit aussi ciblé. Un wiki (pages d'information collaboratives) est également disponible[29], de même qu'un canal de discussion IRC (Internet Relay Chat)[30]. En cumulant le nombre de téléchargements pour la dernière version de Drools (documentation, Guvnor...), on atteint environs 20000 téléchargements[31].
Drools est souvent cité comme le moteur de règles Open Source le plus populaire. Les développeurs principaux du projet interviennent fréquemment sur le blog et les listes de courriel et apportent ainsi une contribution qualitative à la communauté. Le produit obtient 2 sur 2 pour la taille et la qualité de sa communauté.

CLIPS

CLIPS dispose d'une communauté très compétente techniquement. Plusieurs chercheurs et académiciens s'impliquent dans le projet. Un volume important de publications scientifiques ont traité du produit. Des extensions ont vu le jour dans des domaines aussi variés que la logique floue, les flux de règles, la programmation parallèle ou encore les

---

28. http ://blog.athico.com/

29. http ://community.jboss.org/wiki/JBossRules

30. canal #drools sur le serveur irc.codehaus.org

31. http ://www.jboss.org/drools/downloads.html

outils multi-agents.

Depuis le 22 janvier 2008, près de 100000 téléchargements de CLIPS ont été enregistrés sur le site SourceForge[32]. Ce chiffre ne comptabilise pas les installations de paquetages fournis avec les distributions Linux.

Actuellement un groupe sur *Google Groups* semble réunir la communauté CLIPS la plus importante. 1824 membres y sont enregistrés et plusieurs messages y sont publiés tous les mois. Pendant le mois d'avril 2010, 47 messages ont été échangés, ce qui est considérable pour un produit aussi spécialisé. Certains mois, les échanges ont dépassé la centaine de messages.

Le projet est également cité dans de nombreux cours universitaires[33][34][35], tutoriels[36] et présentations sur les systèmes experts.

Ainsi, CLIPS jouit d'une communauté importante et de qualité, composée de chercheurs, d'académiciens et de développeurs professionnels. De plus, le produit affiche un volume conséquent de téléchargements. Il obtient la note de 2 sur 2.

OpenRules

OpenRules ne dispose pas de forums, ni de liste de courriel publique (mailing list). Le modèle de communication est très fermé : les communications se font exclusivement par courriel avec l'entreprise ou en achetant un service d'assistance professionnel payant. Toutes les décisions sont prises en interne dans la société OpenRules Inc.

Le produit ne satisfait aucunement le critère et obtient une note de 0 sur 2.

---

32. http ://sourceforge.net/projects/clipsrules/files/CLIPS/

33. http ://cstl-csm.semo.edu/zhang/CS490_2010/protected/ClassFiles/AI9B.ppt

34. http ://pesona.mmu.edu.my/ ytbau/tes3211/expert_systems_clips.html

35. http ://users.encs.concordia.ca/ andreev/CLIPS_and_JESS.ppt

36. http ://iweb.tntech.edu/bhuguenard/ds6530/ClipsTutorial/tableOfContents.htm

5.10.2 Publications

Drools

La version 4.0.7 de Drools a été publiée en mai 2008. La mise à jour majeure 5.0 est sortie un an plus tard. Une mise à jour mineure 5.1 est actuellement en développement. Des instantanés de la version en développement sont régulièrement publiés pour être évalués par la communauté qui produit un retour d'information précieux. Si l'on considère les statistiques les plus récentes, on peut estimer qu'une mise à jour par an constitue une bonne fréquence de publication. En effet, le moteur de règles est un composant critique dans le système informatique d'une entreprise. Ses fonctionnalités (évaluation de règles) et son API sont stables et il n'est pas sensé être modifié tous les deux ou trois mois. Par ailleurs, plusieurs petites fonctionnalités et projets parallèles sont régulièrement publiés et incorporés aux versions en développement (module Scala pour Drools, modification des algorithmes du moteur de règles...). Drools obtient donc 2 sur 2 pour ce critère.

CLIPS

La première version de CLIPS est apparue en 1985. La version 3.0 a été publiée à l'été 1986. Des améliorations ont suivi, transformant le produit d'un outil d'expérimentation en un outil de développement de systèmes experts. Les versions 4.0 et 4.1, parues en 1987 ont largement amélioré les performances tout en rajoutant la possibilité d'intégration aux langages externes. Un an plus tard, la version 4.2 fût une réécriture complète afin de directement prendre en charge la modularité. La version 4.3 est parue en 1989. Le produit était désormais stable et mature. En 1991, une mise à jour majeure (CLIPS 5.0) introduit la programmation procédurale (fonctions...) et orientée objet (classes, héritage...). La même année, la version 5.1 constituait une mise à jour mineure. Les versions 6.0 et 6.1 sont apparues respectivement en 1993 et en 1998. En 2002, CLIPS 6.20 fût publié. La dernière version stable de CLIPS est la 6.24, parue en 2006. Une mise à jour 6.30 est en cours de développement et une version beta est actuellement téléchargeable. Tout au long de son existence, le produit a connu, en plus des projets

et des extensions annexes, plusieurs publications, apportant souvent des améliorations considérables. Il obtient donc la note de 2 sur 2.

OpenRules

La version 1.0.0 d'OpenRules a vu le jour en décembre 2003, suivie des versions 1.0.1 et 1.0.2, respectivement 4 et 5 mois plus tard. En novembre 2004, la version 1.1.0 a principalement apporté la détection d'erreurs dans les tableurs via le plugiciel *eclipse*.
La version 2.0.0, sortie en mars 2005 a ajouté la création de formulaires web ainsi que l'exposition des projets OpenRules comme services web. Plusieurs mises à jour mineures ont suivi en mai (2.1.0), septembre (2.2.0), novembre 2005 (2.2.1) et en février 2006.
En septembre 2006, la version 3.0.0 est apparue avec notamment un plugiciel *eclipse* et un mécanisme de génération de services web améliorés.
L'utilisation d'expressions du langage naturel, les dépôts de bases de données et l'utilisation de code Java dans les cellules de données des tables décisionnelles sont arrivés avec la version 4.0.0, parue en juillet 2007 et suivie par une mise à jour mineure en septembre 2007.
La version 5.0.1, qui est actuellement payante, a introduit les modèles de règles. Six mises à jour mineures ont suivi, à la fréquence d'une publication tous les quatre mois. La version la plus récente est la 5.3.2, parue en octobre 2009. Elle apporte un assistant de création de questionnaires web basés sur les règles.
OpenRules présente donc un rythme de publication rapide, avec l'ajout de fonctionnalités importantes à chaque mise à jour majeure. Le produit obtient la note de 2 sur 2.

5.10.3 Longévité

Drools

Le projet Drools a été initié en 2001 par Bob McWhirter. Il fut d'abord hébergé sur Sourceforge puis sur Codehaus. En 2005, le produit est racheté par JBoss. En 2006, Red

Hat rachète JBoss et prend en charge le financement du projet. Ainsi, Drools existe depuis près de 10 ans. Dès la version 2.0, le produit s'est imposé comme le moteur de règles Java à code ouvert le plus populaire [37].
Drools est donc un système mature créé depuis longtemps et régulièrement mis à jour. Il obtient 2 sur 2 pour ce critère.

CLIPS

Les origines de CLIPS remontent à 1984. La section d'intelligence artificielle de la NASA avait alors développé une douzaine de prototypes de systèmes experts. La première version fût réalisée en 1985. Le produit a atteint sa version 3.0 en 1986. Pendant trois décennies, CLIPS a été régulièrement utilisé et amélioré. Au fil du temps, il a bénéficié de plusieurs ajouts tel que la programmation objet et l'intégration à des langages externes. Les performances ont également été améliorées d'une version à l'autre. À ce jour, une communauté active continue de développer le produit et de publier de nouvelles versions. Ainsi, CLIPS est un produit mature et non obsolète. Il obtient la note de 2 sur 2.

OpenRules

La société OpenRules, Inc. a été fondée en février 2003. Le nom original de la compagnie était "Intelligent ChoicePoint, Inc." avant de devenir "OpenRules, Inc." en décembre 2003, à la publication de la première version d'OpenRules.
Depuis, sept années se sont écoulées. Le produit a gardé les mêmes fondements mais s'est étoffé de plusieurs fonctionnalités. OpenRules est toujours activement développé. On peut considérer que le produit est mature et non obsolète. On lui attribue une note de 2 sur 2 pour ce critère.

37. http ://en.wikipedia.org/wiki/Drools

5.10.4 Licence

Drools

Drools utilise la licence Apache. Il s'agit d'une licence de logiciel libre créée par la fondation ASF (Apache Software Foundation). La licence requiert la préservation de la notification sur les droits d'auteurs. Le code source sous licence ASF peut être réutilisé pour développer des logiciels soit propriétaires soit libres et à code ouvert. La fondation Apache et la Fondation de logiciel libre (Free Software Foundation) s'accordent à considérer la licence Apache comme une licence de logiciel libre, compatible avec la version 3 de la licence GPL [38].
Ainsi, Drools utilise une licence en accord avec les exigences de la Chaire et obtient une note de 2 sur 2.

CLIPS

CLIPS était la propriété de l'agence aérospaciale américaine NASA, mais il appartient désormais au domaine public. Toute personne est donc libre d'obtenir une copie gratuite du logiciel et des fichiers de documentations qui lui sont associés, de l'utiliser, le copier, le modifier, le distribuer et même en vendre des copies. Les personnes faisant une utilisation commerciale du produit ou désirant supporter les développeurs peuvent envoyer des donations.
CLIPS est en accord avec les exigences de la Chaire et obtient une note de 2 sur 2.

OpenRules

OpenRules propose une licence libre et une licence propriétaire :
- Les projets à code ouvert, utilisant une licence GPL ou une licence compatible, peuvent utiliser OpenRules sous licence GNU General Public License (GPLv2). GPL requiert que tout travail dérivé porte la même licence.

38. http ://www.gnu.org/licenses/license-list.html#apache2

- Les projets à code propriétaire ou utilisant une licence incompatible avec GPL doivent acquérir une licence non GPL d'OpenRules. La licence coûte actuellement 2,995.00 $ US et n'est valable que pour une seule installation du produit. Les organismes à but non commercial, les universités et les sociétés de développement naissantes (*startups*) qui ne veulent pas ouvrir le code de leurs projets ou qui ont des ressources financières limitées peuvent déposer une demande à la société OpenRules pour une licence commerciale à frais réduits.

Dans les deux cas, le téléchargement de la dernière version OpenRules 5 est chargé 29.95$ US.

Le fait d'obliger les projets à code ouvert à payer pour obtenir la version la plus récente et l'existence d'une licence propriétaire font qu'OpenRules ne cadre pas totalement avec les exigences de la Chaire sur ce point. Le produit obtient une note de 1 sur 2.

5.10.5 Assistance communautaire

Drools

Comme il est mentionné au critère *5.10.1 Communauté*, Drools offre des listes de courriel[39] dont une réservée aux questions des utilisateurs. Les développeurs du projet y répondent quotidiennement et de manière détaillée. Par ailleurs, le produit dispose d'un logiciel de suivi des bogues (bug tracker)[40]. Les notifications de bogues y sont répertoriées par modules Drools. Une notification validée se voit attribuer une priorité de résolution et est attribuée à un développeur du projet. Tout utilisateur enregistré est capable de faire remonter des bogues, de suivre la résolution d'un bogue en particulier ou de commenter un bogue précédemment rapporté.

Ainsi Drools dispose d'une assistance communautaire de qualité et d'une plate-forme publique de traitement des défaillances. Il obtient donc 2 sur 2 pour ce critère.

---

39. http ://www.jboss.org/drools/lists.html

40. https ://jira.jboss.org/jira/browse/JBRULES

CLIPS

La documentation officielle (CLIPS User Guide) de 2006 indique des forums et des listes de courriels CLIPS. Ces sites sont inaccessibles actuellement. Toutefois, il existe une communauté active sur Google Groups. N'importe qui, possédant un compte Google, est capable de s'inscrire au groupe et de bénéficier de l'aide de la communauté ou contribuer, selon son niveau d'expertise. Le groupe possède trois ou quatre membres qui représentent les développeurs actuellement en charge du projet. Depuis que le groupe existe (2008), ces membres ont communiqué plus de 600 fois, parfois pour faire des annonces, mais très souvent pour venir en aide aux utilisateurs du produit. Dans le cadre de l'évaluation, on a posé des questions sur le forum et on a obtenu des réponses dans un délais de 48 heures.
Ainsi CLIPS fournit une assistance communautaire rapide et de qualité. Il obtient une note de 2 sur 2.

OpenRules

Le produit OpenRules ne dispose pas d'un forum communautaire, ni d'une liste de courriel ouverte, ni d'un logiciel de suivi de bogues disponible au public. Il est toutefois possible d'envoyer des courriels au service d'assistance technique d'OpenRules. Les utilisateurs qui ont téléchargé une version gratuite antérieure à la version 5.x ont la priorité la plus faible pour traiter leurs questions. Dans le cadre de notre évaluation, nous avons envoyé une question à l'assistance technique et nous avons obtenu une réponse dans un délai de deux semaines.
Ainsi, l'assistance communautaire est limitée au strict minimum. Le produit obtient une note de 1 sur 2 sur ce critère.

5.10.6 Assistance professionnelle

Drools

JBoss est une filiale de Red Hat. Ceci signifie que Drools est financé et développé par une compagnie majeure du secteur informatique. Une version commerciale de Drools, nommée JBoss Rules, est d'ailleurs proposée par Red Hat. Cette version offre un service d'assistance destiné aux entreprises, avec un suivi à long terme et une stabilité accrue du produit (backport des nouvelles fonctionnalités vers d'anciennes versions du produit). Drools jouit donc du soutien d'une compagnie majeure et offre un service d'assistance destiné aux entreprises. Il obtient la note de 2 sur 2.

CLIPS

CLIPS fût conçu et financé par la NASA. Dans le cadre de leurs besoins, plusieurs entreprises ont également fourni des extensions au produit. Actuellement, CLIPS appartient au domaine public. Sa maintenance et son développement reposent sur une communauté de développeurs et d'académiciens bénévoles. CLIPS ne propose donc pas de service officiel d'assistance aux entreprises. La note pour l'assistance professionnelle est de 0.

OpenRules

OpenRules, Inc. est l'entreprise qui développe OpenRules et qui fournit une assistance technique et des services de consultation liés au produit.
Les utilisateurs ayant payé pour la version la plus récente ont droit à un traitement prioritaire de leurs questions par courriel.
OpenRules propose un service d'assistance technique annuel dont le prix varie selon le niveau d'assistance. Actuellement, 6 incidents, 6 mois de mise à jour et une assistance téléphonique limitée sont facturés à 995.00 $ US. 30 incidents, 12 mois de mise à jour et une assistance téléphonique étendue coûtent 2,995.00 $ US.

On pense que ces tarifs sont excessifs pour un produit qui réutilise largement des technologies existantes et libres et qui n'offre pas une grande valeur ajoutée. L'entreprise semble cibler des utilisateurs métier non compétents en informatique, et leur vendre l'expertise informatique.

Le produit obtient une note de 1 sur 2.

5.11 Résultats des évaluations

5.11.1 Scénario 1

Attribut	Poids	Dr	Cl	OR
A. Critères fonctionnels				
Langage de définition des règles				
1. Prise en charge de plusieurs moyens d'expression des règles	2	2	1	0
2. Expressivité du langage de définition des règles	5	2	1	1
3. Prise en charge de l'internationalisation	4	1	0	1
4. Compréhension du modèle de données	5	2	1	2
Fonctions facilitant l'expression des règles				
5. Modèles de règles	4	1	0	1
6. Procédures et algorithmes	4	2	2	2
7. Flux de règles	3	2	2	1
8. Tables et arbres de décision	3	2	0	2
Accessibilité aux experts métier				
9. Accessibilité du langage aux non informaticiens	3	2	1	2
10. Abstraction des détails d'implémentation	3	2	0	2
11. Interfaces pour non informaticiens	3	2	1	2
Le moteur de règles				
12. Stratégies du moteur d'inférence	5	1	2	0
13. Explication du raisonnement et gestion de l'incertitude	4	1	1	0
14. Intégration à l'environnement	3	2	1	2

15. Changements dynamiques	4	1	2	0
Outils de vérification des règles				
16. Vérification de la syntaxe des règles	4	2	2	1
17. Stratégies de résolution de conflit	5	2	1	0
18. Débogage	4	2	2	1
19. Test des règles	4	2	2	2
20. Journalisation et audit	4	2	1	1
Le stockage des règles				
21. Versionnage des règles	5	2	0	1
22. Hiérarchisation des règles	4	2	1	2
23. Définition de méta-données	3	1	0	1
24. Recherche par critères	5	1	1	1
25. Administration des dépôts	3	2	0	1
B. Critères non fonctionnels				
Performance et coût				
26. Performance des moteurs de règles	5	1	2	1
27. Coûts en ressources	3	1	2	1
Prise en main				
28. Documentation	4	2	2	2
29. Convivialité	3	2	1	2
30. Courbe d'apprentissage	4	2	2	2
Évaluation sur l'aspect logiciel libre				
31. Communauté	4	2	2	0
32. Publications	4	2	2	2
33. Longévité	4	2	2	2
34. Licence	5	2	2	1
35. Assistance communautaire	4	2	2	1
36. Assistance professionnelle	3	2	0	1

Totaux pour le scénario 1				
Totaux	278	241	176	166
Pourcentage	100	87	63	60

Tableau 5.1: Évaluation des moteurs de règles selon S1

5.11.2 Scénario 2

Attribut	Poids	Dr	Cl	OR
A. Critères fonctionnels				
Langage de définition des règles				
1. Prise en charge de plusieurs moyens d'expression des règles	4	2	1	0
2. Expressivité du langage de définition des règles	5	2	1	1
3. Prise en charge de l'internationalisation	5	1	0	1
4. Compréhension du modèle de données	5	2	1	2
Fonctions facilitant l'expression des règles				
5. Modèles de règles	4	1	0	1
6. Procédures et algorithmes	4	2	2	2
7. Flux de règles	4	2	2	1
8. Tables et arbres de décision	4	2	0	2
Accessibilité aux experts métier				
9. Accessibilité du langage aux non informaticiens	4	2	1	2
10. Abstraction des détails d'implémentation	3	2	0	2
11. Interfaces pour non informaticiens	4	2	1	2
Le moteur de règles				
12. Stratégies du moteur d'inférence	4	1	2	0
13. Explication du raisonnement et gestion de l'incertitude	2	1	1	0

14. Intégration à l'environnement	3	2	1	2
15. Changements dynamiques	2	1	2	0
Outils de vérification des règles				
16. Vérification de la syntaxe des règles	4	2	2	1
17. Stratégies de résolution de conflit	5	2	1	0
18. Débogage	3	2	2	1
19. Test des règles	3	2	2	2
20. Journalisation et audit	4	2	1	1
Le stockage des règles				
21. Versionnage des règles	4	2	0	1
22. Hiérarchisation des règles	5	2	1	2
23. Définition de méta-données	3	1	0	1
24. Recherche par critères	4	1	1	1
25. Administration des dépôts	3	2	0	1
B. Critères non fonctionnels				
Performance et coût				
26. Performance des moteurs de règles	4	1	2	1
27. Coûts en ressources	3	1	2	1
Prise en main				
28. Documentation	4	2	2	2
29. Convivialité	3	2	1	2
30. Courbe d'apprentissage	4	2	2	2
Évaluation sur l'aspect logiciel libre				
31. Communauté	4	2	2	0
32. Publications	4	2	2	2
33. Longévité	4	2	2	2
34. Licence	5	2	2	1
35. Assistance communautaire	4	2	2	1

36. Assistance professionnelle	3	2	0	1
Totaux pour le scénario 2				
Totaux	270	243	168	170
Pourcentage	100	90	62	63

Tableau 5.2: Évaluation des moteurs de règles selon S2

5.11.3 Scénario 3

Attribut	Poids	Dr	Cl	OR
A. Critères fonctionnels				
Langage de définition des règles				
1. Prise en charge de plusieurs moyens d'expression des règles	2	2	1	0
2. Expressivité du langage de définition des règles	5	2	1	1
3. Prise en charge de l'internationalisation	3	1	0	1
4. Compréhension du modèle de données	5	2	1	2
Fonctions facilitant l'expression des règles				
5. Modèles de règles	3	1	0	1
6. Procédures et algorithmes	4	2	2	2
7. Flux de règles	3	2	2	1
8. Tables et arbres de décision	4	2	0	2
Accessibilité aux experts métier				
9. Accessibilité du langage aux non informaticiens	2	2	1	2
10. Abstraction des détails d'implémentation	2	2	0	2
11. Interfaces pour non informaticiens	2	2	1	2
Le moteur de règles				
12. Stratégies du moteur d'inférence	3	1	2	0

13. Explication du raisonnement et gestion de l'incertitude	4	1	1	0
14. Intégration à l'environnement	5	2	1	2
15. Changements dynamiques	3	1	2	0
Outils de vérification des règles				
16. Vérification de la syntaxe des règles	4	2	2	1
17. Stratégies de résolution de conflit	3	2	1	0
18. Débogage	5	2	2	1
19. Test des règles	5	2	2	2
20. Journalisation et audit	4	2	1	1
Le stockage des règles				
21. Versionnage des règles	3	2	0	1
22. Hiérarchisation des règles	3	2	1	2
23. Définition de méta-données	3	1	0	1
24. Recherche par critères	3	1	1	1
25. Administration des dépôts	3	2	0	1
B. Critères non fonctionnels				
Performance et coût				
26. Performance des moteurs de règles	4	1	2	1
27. Coûts en ressources	2	1	2	1
Prise en main				
28. Documentation	4	2	2	2
29. Convivialité	2	2	1	2
30. Courbe d'apprentissage	4	2	2	2
Évaluation sur l'aspect logiciel libre				
31. Communauté	4	2	2	0
32. Publications	4	2	2	2
33. Longévité	4	2	2	2
34. Licence	5	2	2	1

35. Assistance communautaire	3	2	2	1
36. Assistance professionnelle	2	2	0	1
Totaux pour le scénario 3				
Totaux	248	220	162	155
Pourcentage	100	89	65	62

Tableau 5.3: Évaluation des moteurs de règles selon S3

5.11.4 Scénario 4

Attribut	Poids	Dr	Cl	OR
A. Critères fonctionnels				
Langage de définition des règles				
1. Prise en charge de plusieurs moyens d'expression des règles	3	2	1	0
2. Expressivité du langage de définition des règles	5	2	1	1
3. Prise en charge de l'internationalisation	3	1	0	1
4. Compréhension du modèle de données	5	2	1	2
Fonctions facilitant l'expression des règles				
5. Modèles de règles	4	1	0	1
6. Procédures et algorithmes	4	2	2	2
7. Flux de règles	3	2	2	1
8. Tables et arbres de décision	4	2	0	2
Accessibilité aux experts métier				
9. Accessibilité du langage aux non informaticiens	5	2	1	2
10. Abstraction des détails d'implémentation	5	2	0	2
11. Interfaces pour non informaticiens	5	2	1	2
Le moteur de règles				

12. Stratégies du moteur d'inférence	3	1	2	0
13. Explication du raisonnement et gestion de l'incertitude	2	1	1	0
14. Intégration à l'environnement	4	2	1	2
15. Changements dynamiques	3	1	2	0
Outils de vérification des règles				
16. Vérification de la syntaxe des règles	5	2	2	1
17. Stratégies de résolution de conflit	5	2	1	0
18. Débogage	3	2	2	1
19. Test des règles	3	2	2	2
20. Journalisation et audit	4	2	1	1
Le stockage des règles				
21. Versionnage des règles	4	2	0	1
22. Hiérarchisation des règles	3	2	1	2
23. Définition de méta-données	3	1	0	1
24. Recherche par critères	4	1	1	1
25. Administration des dépôts	3	2	0	1
B. Critères non fonctionnels				
Performance et coût				
26. Performance des moteurs de règles	3	1	2	1
27. Coûts en ressources	5	1	2	1
Prise en main				
28. Documentation	5	2	2	2
29. Convivialité	5	2	1	2
30. Courbe d'apprentissage	5	2	2	2
Évaluation sur l'aspect logiciel libre				
31. Communauté	5	2	2	0
32. Publications	4	2	2	2
33. Longévité	4	2	2	2

34. Licence	5	2	2	1
35. Assistance communautaire	5	2	2	1
36. Assistance professionnelle	5	2	0	1
Totaux pour le scénario 4				
Totaux	292	262	180	186
Pourcentage	100	90	62	64

Tableau 5.4: Évaluation des moteurs de règles selon S4

5.11.5 Bilan

On constate que la solution Drools arrive largement en tête pour les quatre scénarios. Le produit satisfait totalement la majorité des critères et ses scores approchent de la note maximale.

CLIPS devance OpenRules lorsque les règles sont volumineuses et sophistiquées et lorsqu'il existe des compétences informatiques internes importantes. Ceci est dû au moteur plus sophistiqué et performant que celui d'OpenRules, aux outils de test et de débogage offerts, ainsi qu'à la possibilité d'expliquer (partiellement) le raisonnement.

Quant à OpenRules, il obtient une meilleure note que CLIPS lorsque les utilisateurs ont des profils différents ou quand l'implication des experts métier est importante. L'avance d'OpenRules s'explique par une meilleure prise en charge de l'internationalisation, l'utilisation de tables décisionnelles, la hiérarchisation des règles dans des dépôts et une prise en main aisée.

Dans les quatre scénarios, l'écart est très faible entre CLIPS et OpenRules ; chaque produit se distinguant pour une partie des critères, mais échouant pour les autres.

Ce chapitre nous a permis de produire une évaluation exhaustive de trois systèmes de gestion de règles libres, à savoir Drools, CLIPS et OpenRules. Une analyse critère par

critère, conclue par l'attribution d'une note de 0 à 2 pour chaque produit, nous a permis de dégager les forces et les faiblesses des systèmes évalués. Les notes obtenues ont été pondérées selon les quatre scénarios définis précédemment, dans le but d'apporter un contexte d'évaluation concret.

Le chapitre *VII. Solution Adoptée* interprétera en détail les résultats obtenus, afin de produire une recommandation pour la Chaire. La recommandation portera également sur les moteurs de flux ; leur évaluation est détaillée dans le chapitre suivant.

CHAPITRE VI

ÉVALUATION DES MOTEURS DE FLUX DE TRAVAIL

L'évaluation des moteurs de flux de travail se fera selon les critères établis au chapitre IV. Le système de notation est le même que celui des moteurs de règles, à savoir une note de zéro à deux pour chaque critère, en fonction du niveau de satisfaction. Des poids sont ensuite attribués aux critères selon les quatre scénarios définis précédemment. Enfin, un bilan global de l'évaluation est dressé.

Parmi les critères d'évaluation, certains sont une synthèse des patrons de flux de travail de la Workflow Patterns Intitiative. Étant donné que nous n'évaluons pas la conformité des moteurs de flux aux patrons, mais que nous évaluons leur pertinence par rapport aux organismes de finance sociale et solidaire, les patrons sont regroupés par thèmes et une note globale est attribuée pour chaque thème. Si le produit permet d'exprimer directement les patrons liés à un thème (par exemple les contrôles de flux avancés), il obtiendra une note de 2. Si certains ou tous les patrons nécessitent un effort de programmation peu considérable ou si quelques patrons ne sont pas du tout pris en charge, nous attribuerons une note de 1. Si plusieurs patrons ne sont pas pris en charge ou nécessitent un effort de programmation considérable, le produit obtient la note de 0.

6.1 Présentation des produits à évaluer

Comme pour les moteurs de règles, nous effectuons une pré-sélection des moteurs de flux de travail selon leur degré d'activité, l'importance de leurs communautés respectives et leur degré d'aboutissement en vue d'une utilisation dans un environnement de

production.

Parmi les produits écartés de l'étude, on cite :

- Intalio BPM Community Edition [1] qui utilise un serveur Apache Geronimo et Apache ODE ainsi que l'éditeur graphique de diagrammes BPMN d'*eclipse*. Le produit semble intéressant, cependant son code est fermé et sa licence est propriétaire et très restrictive : Le serveur d'application à utiliser est imposé par la licence et il est indiqué que la société Intalio se réserve le droit de changer les conditions d'utilisation à tout moment.
- YAWL (Yet Another Workflow Language) [2] est un produit auquel ont contribué certains des créateurs de la Workflow Patterns Initiative [3]. Le produit est destiné plus à un usage académique qu'à un usage en entreprise. Son principal objectif est d'implémenter 'directement' les divers patrons de flux de travail.
- Ruote [4] apporte une approche innovante, qui se base sur un DSL Ruby pour définir et gérer les flux de travail. Cependant, aucune utilisation du produit en entreprise n'est à signaler. Le moteur est défini par son créateur comme 'simpliste et naïf' [5] et n'est pas conforme aux standards de définition des processus établis dans l'industrie. L'auteur souligne également que Ruote est recommandé pour du prototypage et de l'expérimentation plutôt que pour une utilisation en entreprise [6].
- WfMOpen [7] est une implémentation Java EE d'un moteur de flux de travail tel que

---

1. http ://www.intalio.com/products/bpm/ (dernière consultation le 09-11-10)

2. http ://www.yawl-system.com/ (dernière consultation le 09-11-10)

3. http ://www.workflowpatterns.com/ (dernière consultation le 09-11-10)

4. http ://openwferu.rubyforge.org/ (dernière consultation le 09-11-10)

5. http ://www.slideshare.net/jmettraux/ruote-presentation-in-stockholm-presentation, slide 39 (dernière consultation le 09-11-10)

6. http ://www.slideshare.net/jmettraux/openwferu-ruote-bpminna-june-2008, slide 12 (dernière consultation le 09-11-10)

7. http ://wfmopen.sourceforge.net (dernière consultation le 09-11-10)

spécifié par l'organisme WfMC et l'Object Management Group. Les interfaces de base sont conformes à la spécification OMG[8], elles introduisent quelques modifications pour adapter le service CORBA aux pratiques de conception établies pour une API Java. Les flux de travail sont spécifiés avec une version étendue du langage XPDL de WfMC. Le produit a été écarté de la sélection car (Garces et al., 2009) indique que « l'information fournie dans la documentation est déroutante, dispersée à travers le document et insuffisante à de nombreux égards [...] De plus, l'environnement web de gestion offert est assez difficile d'accès (*unfriendly*) ». Étant donné que la convivialité et l'accessibilité sont des critères importants dans notre évaluation, WfMOpen ne semble pas être le candidat idéal.

Les produits retenus sont tous trois activement développés (dernières versions récentes, nombre conséquent de téléchargements, communauté importante...), utilisables dans un environnement de production et conformes à au moins l'un des standards établis pour la description et l'exécution des processus d'affaires (BPMN, XPDL ou BPEL).

6.1.1 jBPM[9]

jBPM est une plateforme d'exécution de langages de processus. Actuellement, les langages maintenus sont le standard BPMN 2.0 et le langage jPDL (jBPM Process Definition Language) spécifique au produit.

jBPM permet aux utilisateurs métier de schématiser graphiquement les activités, transitions et événements du flux de travail. Une API claire et couvrant différents aspects de la gestion des processus permet aux développeurs d'étendre le système et de réaliser des flux de travail sophistiqués.

Plusieurs outils d'aide au développement sont fournis, dont une console d'administration web développée avec GWT, un éditeur graphique de diagrammes de flux avec une

8. http ://www.omg.org/cgi-bin/apps/doc?formal/00-05-02.pdf (dernière consultation le 09-11-10)

9. http ://www.jboss.org/jbossjbpm (dernière consultation le 09-11-10)

version web et une version pour l'environnement de développement *eclipse*.
jPBM jouit d'une large utilisation en entreprise (British Telecom, Atos, SNS Bank...) et est distribué sous licence LGPL.

6.1.2 Bonita Open Solution [10]

Bonita Open Solution est un système de gestion de flux de travail libre et complet. Il est composé de trois modules intégrés : Bonita Studio, Bonita Forms et Bonita User Experience. Bonita Studio est un éditeur basé sur la plateforme *eclipse* qui permet de dessiner des flux de travail conformes à la notation BPMN. Il offre des fonctionnalités avancées tel que la définition des participants aux flux, l'affectation de priorités aux tâches, la définition de variables, la connexion à des systèmes externes ainsi que la définition et la personnalisation des interfaces utilisateur. Bonita Forms génère les interfaces utilisateur requises pour l'échange de données entre participants humains. Enfin, Bonita User Experience fournit une interface similaire à celles des boîtes de courriel. L'interface permet à l'administrateur de superviser les processus et aux participants de consulter les tâches qu'ils doivent effectuer.
Bonita Open Solution constitue donc une solution intuitive et facile d'accès pour les experts métier. Elle est distribuée sous licence GPLv2.

6.1.3 Enhydra Shark [11]

Enhydra Shark est un système de gestion de flux de travail extensible basé sur les spécifications WfMC et le format XPDL. Enhydra utilise l'API ToolAgents du WfMC pour l'exécution côté serveur des activités système. Chaque composant (couche de persistance, gestionnaire d'affectation des processus..) peut être utilisé dans son implémentation de référence ou étendu/remplacé par des modules client. Ainsi, le moteur de flux de travail peut être utilisé comme une simple bibliothèque Java ou encore invoqué en tant que

10. http ://www.bonitasoft.com/ (dernière consultation le 09-11-10)

11. http ://wwww.enhydra.org/workflow/shark/index.html (dernière consultation le 09-11-10)

service web.

Enhydra Shark fournit un éditeur pour la définition graphique des processus ainsi que des applications d'administration destinées aux non informaticiens.

Le produit est distribué sous licence LGPL.

6.2 Langage de définition des processus

6.2.1 Prise en charge de plusieurs moyens d'expression

jBPM

jBPM se base sur une machine virtuelle de processus appelée *Process Virtual Machine* ou PVM. La machine virtuelle ne définit pas l'implémentation des activités des flux de travail. Elle fournit uniquement l'environnement d'exécution et une API d'activités. La PVM oriente le flux d'exécution d'une activité à l'autre et délègue leur comportement aux composants enfichés de type *ActivityBehaviour*. Ainsi, les langages pris en charge par jBPM sont des ensembles d'implémentations de l'interface *ActivityBehaviour* accompagnés d'un analyseur (*parser*). BPMN2, qui est un standard de modélisation des flux de travail et jPDL (jBPM Process Definition Language), qui est le langage conçu dans le cadre de jBPM, sont les deux formats actuellement pris en charge. Ils s'exécutent au sein de la machine virtuelle (PVM) et partagent une même base (persistance, transactions, configuration, constructions élémentaires des processus...). Par conséquent, l'optimisation du système bénéficie aux deux langages.

En plus de prendre en charge deux langages d'expression des processus, jBPM définit plusieurs types de tâches automatiques :

- Code Java : La méthode Java spécifiée en paramètre est exécutée. Par exemple, en jPDL, le code suivant exécute la méthode *calculerTaxes* et récupère la valeur de retour dans la variable de flux appelée *resultat*.

```
<java name="calculerTaxes"
      class="org.banque.ServiceTaxes"
      method="calculerTaxes"
```

```
        var="resultat">
```

L'invocation de méthodes Java est possible avec le format BPMN aussi.

- Langage de script : Une tâche BPMN ou une activité jPDL peut exécuter un script écrit dans n'importe quel langage ayant une implémentation conforme à la spécification JSR-223 (Python, Ruby, Groovy, BeanShell (expressions Java), awk...).
  La tâche BPMN suivante calcule et affiche le double de chacun des entiers passés en paramètre, en utilisant le langage BeanShell.

```
<scriptTask id="scriptTask" name="Script Task" scriptLanguage="bsh">
  <script><![CDATA[
    for(int i=0; i < input.length; i++){
      System.out.println(input[i] + " x 2 = " + (input[i]*2));
    }]]>
  </script>
</scriptTask>
```

- Requête HQL ou SQL : jPDL permet d'exécuter des requêtes HQL (Hibernate Query Language) et SQL (Structured Query Language) et de récupérer les résultats dans des variables du processus.

```
<hql name="compter les activites"
     var="activites"
     unique="true">
  <query>
    select count(*)
    from org.jbpm.pvm.internal.model.ActivityImpl
  </query>
  <transition to="etapeSuivante" />
</hql>
```

- Règle Drools : L'activité crée une session Drools, y injecte des faits et exécute toutes les règles. Les règles modifient ou créent de nouvelles variables dans le cadre du processus, orientant ainsi son flux d'exécution. Par exemple, le code suivant injecte l'instance *compte* comme fait et déclenche l'exécution des règles.

```
<rules name="evaluerDemande">
  <fact var="compte" />
```

```
    <transition to="afficherEvaluation"/>
  </rules>
```

La règle est définie selon la syntaxe DRL de Drools

```
rule "EvaluerDemandeEmprunt"
  when
    compte : banque.modele.Compte( solde > 0 )
  then
    compte.setEligible( true );
end
```

Ainsi, grâce à l'utilisation d'une machine virtuelle de processus, jBPM est capable d'exécuter des processus définis selon divers formats (actuellement BPMN et jPDL). Le système permet également de définir des activités automatiques en Java, avec un langage de script (Python, Groovy, Ruby...), avec les langages de requêtes HQL et SQL ou encore avec des règles d'affaires Drools.
Le produit satisfait pleinement le critère et obtient une note de 2 sur 2.

Bonita Open Solution

Les processus Bonita sont définis à l'aide d'un éditeur graphique puissant et conforme au standard BPMN 2.0. Les définitions sont enregistrées dans des fichiers XML, générés à partir des diagrammes de flux de travail.
La représentation graphique devrait convenir aussi bien aux experts métier qu'aux informaticiens. De plus, le système permet d'importer des définitions de flux XPDL 1.0, BPMN 2.0 (extension .bpmn) et jBPM 3.2.
Les actions liées aux activités du flux de travail peuvent être définies selon une multitude d'approches, soit avec des expressions Groovy ou Java, soit avec des connecteurs. Les expressions Groovy ou Java peuvent être utilisées dans la création de requêtes SQL, dans la définition des données d'un composant de l'interface utilisateur ou pour le libellé et la description d'un élément du flux.
Un éditeur d'expressions bénéficie de l'auto-complétion, de la coloration syntaxique et d'autres assistants de la plate-forme *eclipse*. Les variables de processus et les variables

d'environnement (instance de processus en cours, activité en cours...) sont accessibles depuis l'éditeur. Un ensemble de fonctions Java et Groovy, organisées par catégories (définies par l'utilisateur, relatives à la manipulation des nombres, des chaînes de caractères, etc) est fourni avec l'éditeur d'expressions. Cliquer sur une fonction affiche sa documentation. Double-cliquer dessus l'insère dans l'expression en cours d'édition.

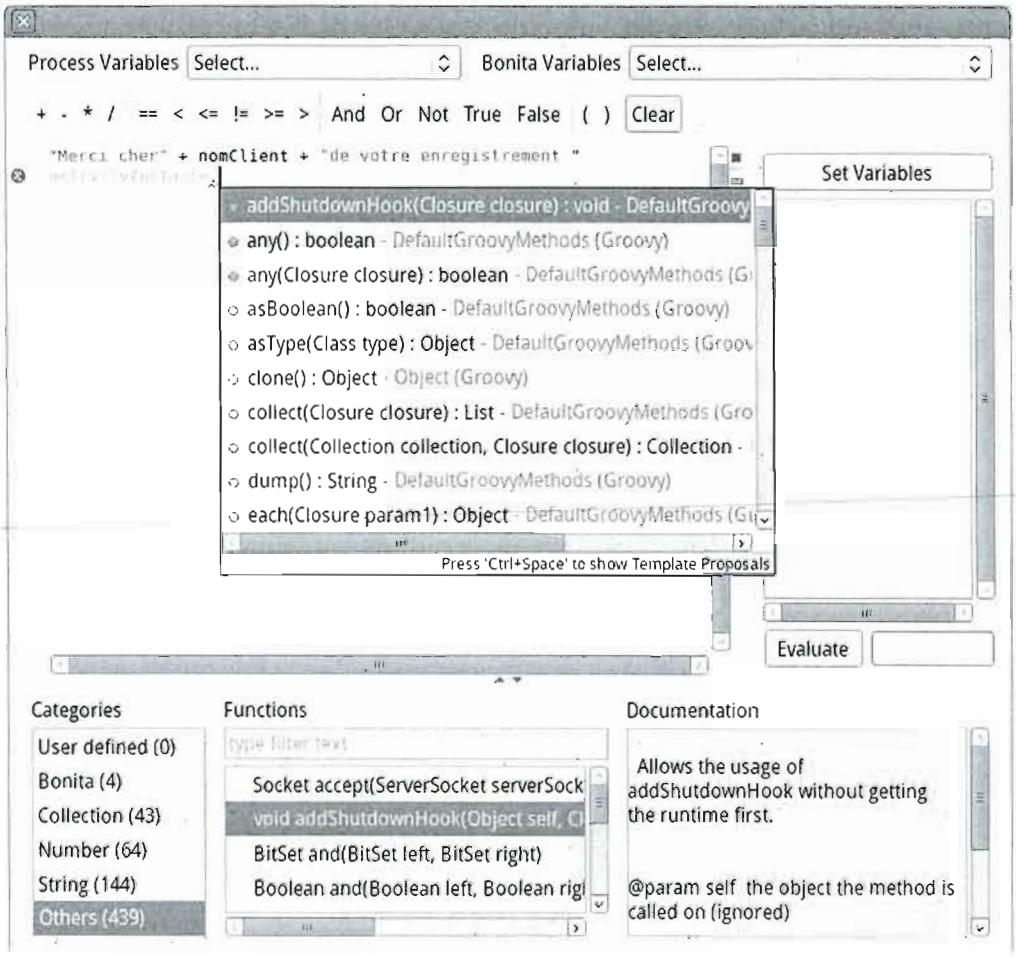


Figure 6.1 Éditeur d'expressions Bonita

En plus de la définition d'expressions Java et Groovy, BOS (Bonita Open Solution) permet d'utiliser des connecteurs. Les connecteurs prennent en entrée des valeurs passées par l'utilisateur et certains connecteurs retournent des résultats au système de gestion

de flux.

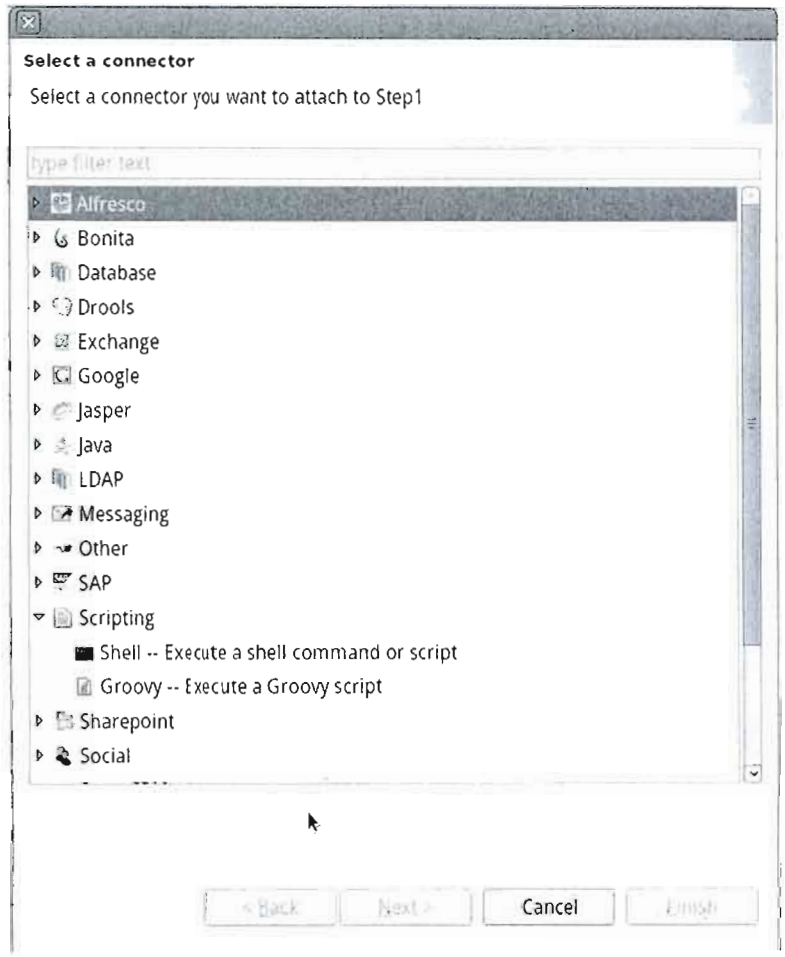


Figure 6.2 Extrait de la liste des connecteurs prédéfinis par Bonita

Pour ajouter un connecteur à une activité, il suffit de cliquer l'activité dans le diagramme et d'aller dans *Details* − > *Connectors* − > *Add*. Il existe toute sorte de connecteurs prédéfinis :

- Connecteur Java : Invoque une méthode Java à partir des classes Java standard ou de classes personnelles importées dans Bonita sous forme d'archives jar. Il est possible de créer son propre connecteur en étendant la classe *ProcessConnector* dont la méthode *executeConnector* définit le code à exécuter par le connecteur. De plus, les langages

de script Python (bibliothèques Jython), Ruby (JRuby) ou encore Groovy peuvent s'exécuter à partir de code Java. La figure 6.3 montre un connecteur personnalisé qui exécute des instructions Python.

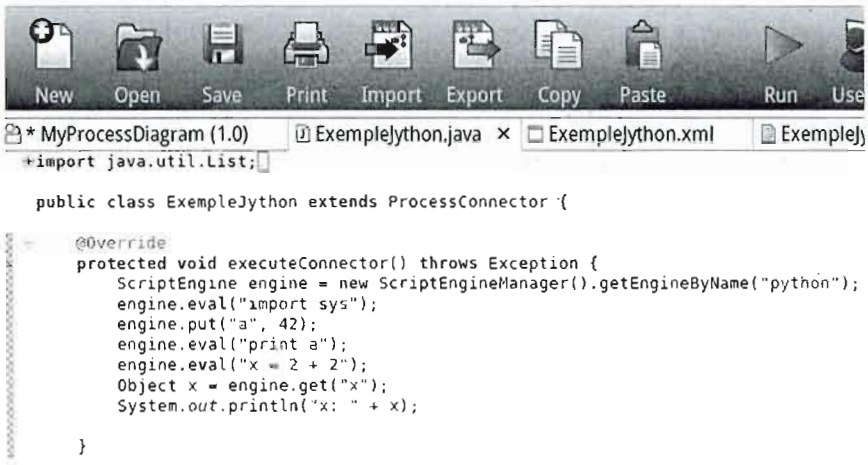


Figure 6.3 Connecteur Java exécutant du code Python

- Connecteur pour bases de données : Plusieurs SGBDs sont pris en charge (Access, PostgreSQL, Sybase, MySQL, DB2, HyperSQL DB, Oracle...). Les données nécessaires pour établir une connexion sont spécifiées dans l'assistant du connecteur.
- Connecteur Drools : Applique des règles Drools pour gérer le flux de travail. Les classes représentant des types de faits sont importées dans des archives .jar. Les faits et les règles Drools à y appliquer sont spécifiés dans l'assistant du connecteur Drools. Les résultats obtenus après avoir exécuté les règles d'affaires sont récupérés dans des variables de processus Bonita.
- Connecteur Services web : Invoque un service web et récupère les résultats de son exécution dans des variables de processus. Les champs nécessaires à l'invocation du service web (nom du port, adresse, protocole, format de données...) sont saisis dans l'assistant du connecteur.
- Connecteur Scripts Shell et Groovy : Exécutent des scripts Shell et Groovy.
- Autres : Partage de documents sur Alfresco, gestion des calendriers Exchange et Google Calendar, recherche dans un répertoire LDAP, envoi de courriel, envoi de

messages sur twitter...

En plus des connecteurs prédéfinis, des connecteurs réalisés par la communauté sont répertoriés sur le site du produit [12] et téléchargeables sous format d'archives .jar directement installables dans Bonita. Il est également possible de créer ses propres connecteurs.

En fonction du type d'élément auquel le connecteur est rattaché, on définit l'étape à laquelle il se déclenche (entrée, démarrage, fin, suspension...) :

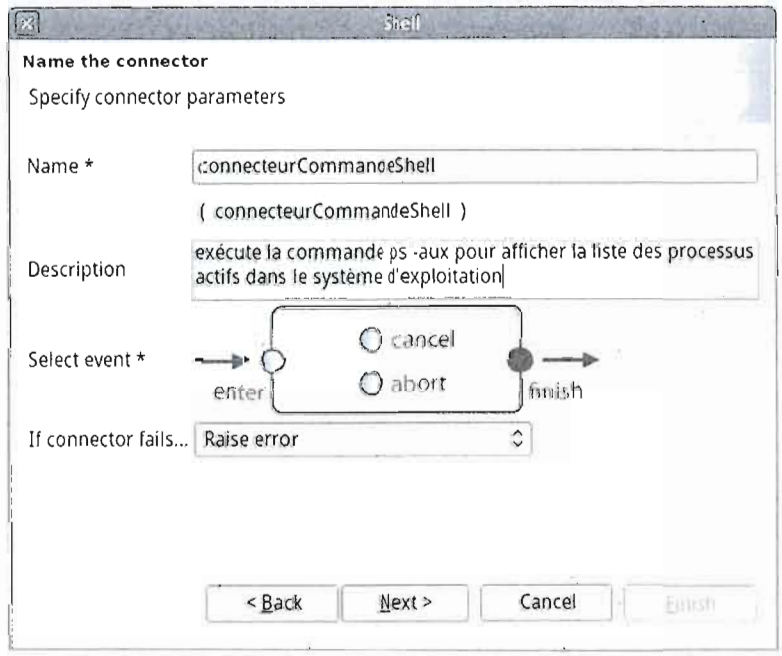


Figure 6.4 Choix de l'étape de déclenchement du connecteur

En définitive, Bonita dispose d'un éditeur graphique ergonomique, puissant et conforme au standard BPMN 2.0. Les définitions de processus peuvent être importées sous format XPDL, jBPM et BPMN (.bpmn). Les valeurs de plusieurs données du système peuvent être saisies sous forme d'expressions Groovy ou Java et les connecteurs permettent d'invoquer une multitude de services externes.

Bonita satisfait amplement ce critère et obtient une note de 2 sur 2.

12. http ://www.bonitasoft.org/exchange/ (dernière consultation le 09-11-10)

Enhydra Shark

Avec Enhydra Shark, les processus sont définis graphiquement, en bénéficiant des assistants d'édition et de modélisation de l'éditeur *Together Workflow Editor* (TWE). Le code XPDL correspondant est automatiquement généré à partir des diagrammes. Les processus peuvent être codés dans un éditeur tiers en manipulant directement du code XPDL. Si les définitions de processus contiennent des extensions propriétaires aux éditeurs XPDL tiers, elles seront ignorées par Enhydra Shark (ES).

D'autre part, des expressions peuvent être formulées dans un langage de script. Au niveau d'un paquetage de processus, l'élément *script* identifie le langage utilisé. ES fournit des interpréteurs pour les langages de script JavaScript, Python et BeanShell.

Concernant la programmation des activités, ES implémente le concept de *Tool Agent* spécifié par la WfMC. Il s'agit d'une interface d'invocation d'applications tierces à partir des processus.

L'utilisateur peut créer son propre *ToolAgent* ou utiliser un agent prédéfini, tel que :

- Agent d'applications *RuntimeApplicationToolAgent* : Exécute des applications externes (éditeur Notepad ou Vi, logiciel de traitement de texte...). L'exécution est soit synchrone (le flux est suspendu jusqu'à ce que l'application termine son exécution) soit asynchrone (démarrer l'application et poursuivre le flux). Un agent d'application peut avoir un alias, ce qui permet de l'invoquer à plusieurs reprises.
- Agent Java *JavaClassToolAgent* : Exécute des classes Java qui implémentent une méthode statique nommée *execute*. L'agent prend en paramètre le nom de la classe à exécuter et les paramètres passés à partir du processus. Un agent Java est capable d'interagir avec toute sorte de bibliothèques du langage. Par exemple, pour utiliser le moteur de règles Drools, il suffit de créer une façade possédant une méthode statique *execute* qui lance une instance sans état (Stateless) du moteur Drools. Les règles et les faits sont passés en paramètre à l'agent.
- Agent BeanShell *BshToolAgent* : Exécute des scripts écrits en Java. Le code suivant définit la valeur de l'attribut *Script* d'un agent BeanShell :

```
<ExtendedAttribute Name="Script" Value="c=new java.lang.Long(a.longValue
```

```
()-b.longValue());"/>
```

- Agent JavaScript *JavaScriptToolAgent* : Exécute du code JavaScript. L'agent prend en paramètre le script à exécuter ou le nom d'un fichier contenant le script. Dans l'exemple suivant, a, b et c sont des variables de processus passées en paramètre à l'agent.

```
<ExtendedAttribute Name="Script" Value="c=a-b;"/>
```

- Agent XSL *XSLTToolAgent* : Applique des transformations XSL (eXtensible Stylesheet Language) à une chaîne de caractères, un tableau d'octets ou une variable XML passée en paramètre et retourne le résultat après transformation.
- Agent services web *SOAPToolAgent* : Exécute des opérations de services web. L'agent prend en paramètre l'emplacement du fichier WSDL qui définit le service web à appeler.
- Agent de courriel *MailToolAgent* : Envoie et reçoit des courriels. Le gestionnaire de courriels implémente l'interface *MailMessageHandler*. Une implémentation par défaut est fournie, mais il est possible de la changer.
- Agent par défaut *DefaultToolAgent* : Lit l'attribut *ToolAgentClass* qui lui est passé en paramètre et appelle l'agent expert approprié.

Pour créer ses propres agents, il suffit d'étendre la classe *AbstractToolAgent* ou l'une de ses implémentations. La sous-classe redéfinit la méthode *invokeApplication*. Les paramètres extraits du processus sont passés dans l'attribut *parameters* de la méthode.

```
public class JavaClassToolAgent extends AbstractToolAgent {
  // Execute l'application
  public void invokeApplication(WMSessionHandle shandle, long handle,
      WMEntity appInfo, WMEntity toolInfo, String applicationName, String
      procInstId, String assId, AppParameter[] parameters, Integer appMode)
...
  public String getInfo(WMSessionHandle shandle) throws
      ToolAgentGeneralException
...
```

La méthode *getInfo* retourne une chaîne de caractères décrivant l'agent.
Ainsi, Enhydra Shark permet de définir les processus de manière graphique, en utilisant l'éditeur de code XPDL fourni ou en utilisant tout éditeur tiers conforme à la spécification XPDL 1.0. Des expressions peuvent être écrites dans un langage de script tel que Python, JavaScript et BeanShell. Enfin, les activités peuvent utiliser toutes sortes d'agents pour exécuter des applications, du code Java ou des services web, entre autres. Le produit obtient la note de 2.

6.2.2 Prise en charge de l'internationalisation

jBPM

jBPM n'offre pas ses propres outils d'internationalisation pour adapter les noms des activités et les messages envoyés aux acteurs humains, entre autres. L'application web de gestion du système de flux *jBPM Console* n'existe qu'en anglais.
Cependant, on peut définir des tâches Java qui se déclenchent automatiquement dans les flux de travail ou pour certaines activités tel que la création de messages à destination d'utilisateurs humains. Ces tâches invoqueraient des méthodes qui adaptent les données à l'environnement local, en utilisant la bibliothèque i18n de Java.
L'application web *jBPM Console* est elle aussi codée en Java avec GWT et peut être modifiée pour utiliser les bibliothèques i18n.
jBPM ne facilite donc pas la gestion de l'internationalisation mais il utilise des technologies qui en permettent une prise en charge avancée. Le produit obtient une note de 1 sur 2.

Bonita Open Solution

Les expressions Java et Groovy définies avec Bonita bénéficient des bibliothèques d'internationalisation i18n de Java. Ainsi, les données échangées dans les processus peuvent s'adapter automatiquement à l'environnement de l'utilisateur.
Par ailleurs, l'application Bonita et l'interface web de gestion des processus User XP

prennent en charge l'internationalisation. Les localisations en français, en anglais et en espagnol sont déjà implémentées. La communauté est encouragée à contribuer la localisation dans d'autres langues.

De plus, l'architecture des connecteurs intègre l'internationalisation et il est très facile d'ajouter la prise en charge d'une langue à partir de l'éditeur Bonita. À titre d'illustra-

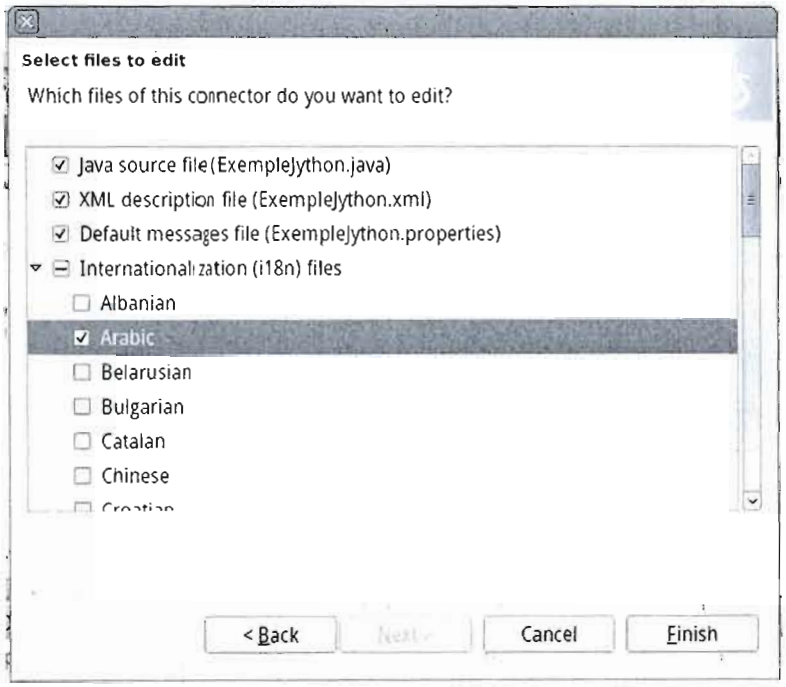


Figure 6.5 Ajout de la prise en charge d'une langue par un connecteur Bonita

tion, l'éditeur génère un fichier *ExempleJython_ar.properties* lorsqu'on désire ajouter la prise en charge de la langue arabe à notre connecteur exécutant du code Python.

Enfin, il est possible d'affecter des libellés écrits en caractères non latins aux éléments du processus tout en leur attribuant des identifiants conformes à l'alphabet ASCII.

Le produit obtient la note de 2 sur 2.

Enhydra Shark

L'interface de l'éditeur de processus *Together Workflow Editor* est disponible en six langues. Dans le fichier de configuration *togwecontroller.properties* de l'éditeur, la propriété *Encoding* définit l'encodage utilisé pour enregistrer les processus (Unicode, Japonais, Russe...). En revanche l'application Swing d'administration des processus et sa version web *sharkWebClient* n'existent qu'en anglais.
Étant une bibliothèque Java, ES est accessible à partir d'applications qui prennent en charge l'internationalisation grâce à la bibliothèque i18n de Java. Ces bibliothèques sont également accessibles aux agents Java utilisés dans les activités.
Par ailleurs, les attributs étendus XPDL peuvent être utilisés pour spécifier les propriétés d'un processus ou de ses éléments selon plusieurs environnements locaux.
Par rapport à Bonita, ES fournit une gestion limitée de l'internationalisation et nécessite un effort de développement. Le produit obtient la note de 1 sur 2.

6.2.3 Compréhension du modèle de données

jBPM

Le modèle de données est accessible dans les méthodes Java déclenchées par des activités ou des événements. En effet, Java fournit plusieurs bibliothèques d'accès aux sources de données externes (Hibernate, JPA...). jBPM bénéficie d'une intégration avancée avec Hibernate puisque les données du système (flux, participants, historique d'exécution...) sont lues et stockées dans des bases de données via l'outil Hibernate.
De plus, les variables de processus sont visibles à toutes les étapes d'exécution d'une instance de processus. Elles sont de type numérique (Integer, Long, Float, Double), caractère ou chaîne de caractères, booléen, tableau d'octets (byte[]), de type entités Hibernate ou de type sérialisable si elles ne correspondent à aucun des types précédents. Si une variable de processus est stockée comme objet sérialisable, il est possible de mettre à jour la version désérialisée sans avoir à ré-enregistrer explicitement l'objet. Les variables désérialisées sont automatiquement mises à jour si elles sont modifiées.

Le code suivant illustre la manipulation d'une variable de processus de type *Client* à partir d'une activité.

```
public class EnregistrerClient implements ActivityBehaviour {
  public void execute(ActivityExecution execution) {
    // Acces a la variable de processus client1
    Client client = (Client) execution.getVariable("client1");
    client.setNom(unNom);
    client.setStatut(Statut.VIP);
  }
}
```

Lorsque la transaction qui invoque l'activité *EnregistrerClient* s'achève, le client est automatiquement mis à jour dans la base de données.
En plus des outils d'accès aux sources de données externes fournis par Java, jBPM permet de définir des variables de processus. Ces variables représentent éventuellement des objets du modèle de données et sont visibles par tous les éléments du processus. La note 2 sur 2 est accordée sur ce critère.

Bonita Open Solution

Les données définies par l'utilisateur sont accessibles dans les processus. Elles sont importées sous forme d'archives de classes Java (.jar). Pour utiliser les archives importées, on sélectionne l'entité (processus, tâche...) où les types complexes seront utilisés et on choisit d'ajouter un objet Java. Les classes contenues dans l'archive importée apparaîtront dans la liste de types sélectionnables.
De plus, l'éditeur Groovy/Java de Bonita permet de modifier les classes importées et d'en créer de nouvelles.
Il est donc aisé d'importer et de manipuler un modèle de données existant à partir des processus Bonita. Le produit obtient la note de 2 sur 2.

La propriété *ExternalReference* permet d'utiliser des classes Java comme variables de processus. L'attribut *location* de la référence externe doit indiquer le nom complet de la classe Java à utiliser. Par exemple, le code suivant définit un champ *compte* de type *banque.domaine.Compte* et un paramètre formel *client* de type *banque.domaine.Client.*

```
<DataField Id="compte" IsArray="FALSE">
    <DataType>
        <ExternalReference location="banque.domaine.Compte"/>
    </DataType>
</DataField>
...
<FormalParameter Id="client" Mode="INOUT">
    <DataType>
        <ExternalReference location="banque.domaine.Client"/>
    </DataType>
</FormalParameter>
```

Alternativement, un élément *TypeDeclaration* peut référencer une classe Java. L'élément peut être utilisé partout dans le processus.

```
<TypeDeclaration Id="type_client">
    <ExternalReference location="org.banque.domaine.Client"/>
</TypeDeclaration>
```

L'utilisation du type déclaré se fait comme suit :

```
<DataField Id="client">
    <DataType>
        <DeclaredType Id="type_client"/>
    </DataType>
</DataField>
```

En plus d'accéder à des types Java dans les processus, il est possible d'interagir avec des sources de données externes en y insérant des données ou en effectuant des requêtes de sélection. Les agents Java peuvent par exemple s'interfacer avec des bases de données

relationnelles via JDBC. Les agents JavaScript et Java sont aussi capables de manipuler des données stockées sous format XML.
Ainsi, les processus définis avec ES peuvent avoir une intégration avancée avec le modèle de données. Le produit obtient la note de 2 sur 2.

6.3 Contrôle du flux de travail

6.3.1 Contrôle élémentaire du flux

jBPM

Étant donné que jBPM permet l'exécution de processus modélisés avec jPDL et BPMN, on évalue la possibilité d'exprimer les contrôles de flux pour les deux langages.

- Séquence de tâches : Pour signifier avec jPDL qu'un élément A s'exécute après un élément B, on place la balise *<transition to="B">* dans la définition de l'élément A. Avec BPMN, la transition est exprimée dans une balise externe aux éléments à relier comme suit :

```
<sequenceFlow id="identifiant" name="nom"
      sourceRef="nom-de-l'element-source"
      targetRef="nom-de-l'element-destination" />
```

- Divergence parallèle et synchronisation : jPDL réalise la divergence parallèle et la synchronisation avec des balises *fork* et *join*.

```
<fork name="fork">
      <transition to="activite parallele 1" />
      <transition to="activite parallele 2"/>
</fork>
...# definition des activites paralleles
<join name="synchronisation" >
      <transition to="activite suivante" />
</join>
```

L'implémentation de BPMN fournie par jBPM utilise des balises *parallelGateway* pour faire diverger et synchroniser des activités parallèles.

```
<parallelGateway id="porte de divergence" name="fork" gatewayDirection="
    diverging"/>
...# definition des taches paralleles
<parallelGateway id="porte de synchronisation" name="synchronisation"
    gatewayDirection="converging"/>
```

- Choix exclusif et convergence simple : jPDL définit les choix exclusifs
  - en liant l'exécution de chaque transition à une expression booléenne

```
<decision name="evaluer document">
    <transition to="soumettre document">
      <condition expr="#{contenu=="acceptable"}" />
    </transition>
...
```

  - en définissant l'expression qui arbitrera le choix exclusif comme attribut de la balise *decision*

```
<decision name="evaluer document" expr="#{contenu}" >
  <transition name="acceptable" to="soumettre document" />
  <transition name="insuffisant" to="refuser document"  />
```

  - en exécutant des règles Drools qui modifient une variable de décision (dans l'exemple suivant, la variable *contenu*)

```
<rules-decision name="contenu">
    <transition name="acceptable" to="soumettre document" />
    <transition name="insuffisant" to="refuser document" />
```

Les chemins alternatifs du flux convergent vers un élément de type *join*.

Avec BPMN, le choix exclusif est réalisé avec des balises *exclusiveGateway* :

```
<exclusiveGateway id="evaluerDocument" name="evaluerDocument" />
<sequenceFlow id="soumettre" name="soumettre"
      sourceRef="evaluerDocument" targetRef="soumettreDocument">
      <conditionExpression xsi:type="tFormalExpression">
        \${contenu == "acceptable"}
      </conditionExpression>
</sequenceFlow>
...
```

Enfin, il est possible de déléguer la décision à une classe Java implémentant l'interface *DecisionHandler*.

jBPM prend en charge les contrôles de flux élémentaires, aussi bien pour le langage jPDL que pour l'implémentation qu'il propose de la notation BPMN. Le produit obtient la note de 2 sur 2.

Bonita Open Solution

Les contrôles élémentaires de flux (séquence, divergence parallèle / synchronisation, choix exclusif / convergence simple) sont directement exprimables avec l'éditeur graphique Bonita, conformément à la norme BPMN et sans avoir à écrire une seule ligne de code. Le produit obtient la note de 2 sur 2.

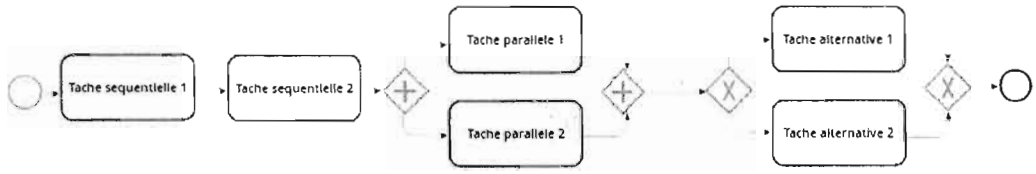


Figure 6.6 Contrôles élémentaires de flux avec Bonita

Enhydra Shark

ES livre des implémentations de certains patrons de flux, accessibles depuis l'éditeur TWE à partir d'un paquetage nommé *Workflow Patterns*. Les implémentations sont accessibles en dehors de l'éditeur, puisqu'il s'agit de fichiers XPDL. Le nom de chaque processus contient l'intitulé et le code attribué au patron implémenté (Par exemple *WfBCP1 Sequence* pour le patron de contrôle de flux de séquence).

- Séquence de tâches : Une suite d'activités est liée par des transitions.
- Divergence parallèle et synchronisation : La divergence parallèle est réalisée avec une contrainte de transition de type *<xpdl :Split Type="AND">*. La synchronisation se fait avec une contrainte *<xpdl :Join Type="AND"/>*.

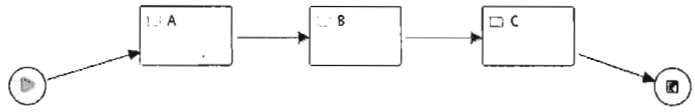


Figure 6.7 Séquence de tâches avec ES

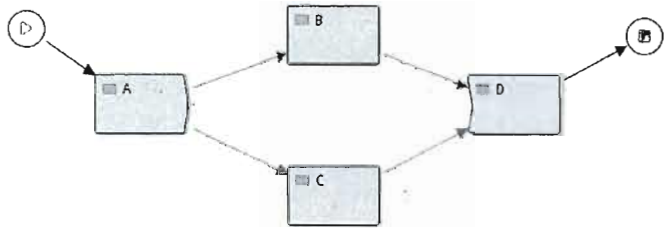


Figure 6.8 Divergence parallèle et synchronisation avec ES

Le code suivant reprend les balises les plus pertinentes du code XPDL sous-jacent.

```
<xpdl:WorkflowProcess Id="Synchronization">
        ...
        <xpdl:Activity Id="A">
        ...
           <xpdl:Split Type="AND">
               <xpdl:TransitionRefs>
                    <xpdl:TransitionRef Id="Tr1" />
                    <xpdl:TransitionRef Id="Tr2" />
        ...
        </xpdl:Activity>

        <xpdl:Activity Id="D">
        ...
             <xpdl:Join Type="AND" />
        </xpdl:Activity>

        <xpdl:Transition Id="Tr1" From="A" To="B" />
        <xpdl:Transition Id="Tr2" From="A" To="C" />
        ...
</xpdl:WorkflowProcess>
```

- Choix exclusif et convergence simple : Réalisé avec une contrainte de type XOR pour les transitions émanant de l'activité de divergence (l'activité *A* dans l'exemple). Un attribut étendu de l'activité définit une variable (appelée *whereToGo* dans l'exemple) dont la valeur désigne la branche à activer à l'exécution.

```
<xpdl:ExtendedAttribute Name="VariableToProcess_UPDATE" Value="whereToGo
    "/>
```

Chaque transition émanant de l'activité *A* porte une condition sur la valeur de la variable *whereToGo*.

```
<xpdl:Transition From="A" Id="Tr1" Name="Transition" To="B">
  <xpdl:Condition Type="CONDITION">whereToGo=="B"</xpdl:Condition>
```

Une condition de type *<xpdl :Condition Type="OTHERWISE"/>* est l'équivalent de *else* en Java. Elle est activée si toutes les autres conditions sont évaluées à faux. Quant à la convergence simple, elle est implémentée avec la contrainte *<xpdl :Join Type="XOR"/>*.

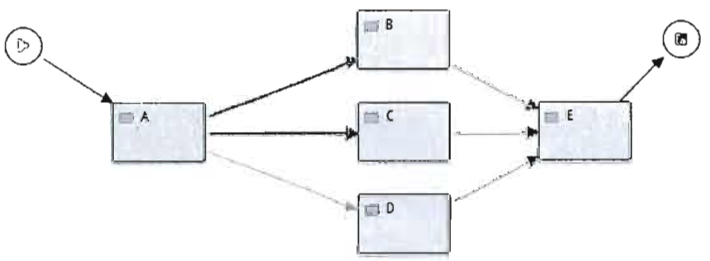


Figure 6.9 Choix exclusif et convergence simple avec ES

Les extraits du code XPDL les plus pertinents sont repris ci-dessous.

```
<xpdl:WorkflowProcess Id="Simple_Merge">
        ...
        <xpdl:Activity Id="A">
        ...
           <xpdl:Split Type="XOR">
               <xpdl:TransitionRefs>
                   <xpdl:TransitionRef Id="Tr1"/>
                   <xpdl:TransitionRef Id="Tr2"/>
```

```
                    <xpdl:TransitionRef Id="Tr3" />
                ...
                <xpdl:ExtendedAttribute Name="VariableToProcess_UPDATE"
                    Value="whereToGo" />
            </xpdl:Activity>
            ...
            <xpdl:Activity Id="E">
            ...
                    <xpdl:Join Type="XOR" />
            </xpdl:Activity>

            <xpdl:Transition Id="Tr1" From="A" To="B">
                <xpdl:Condition Type="CONDITION">whereToGo=="B"</
                    xpdl:Condition>
            </xpdl:Transition>

            <xpdl:Transition Id="Tr2" From="A" To="C">
                <xpdl:Condition Type="CONDITION">whereToGo=="C"</
                    xpdl:Condition>
            </xpdl:Transition>

            <xpdl:Transition Id="Tr3" From="A" To="D">
                <xpdl:Condition Type="OTHERWISE" />
            </xpdl:Transition>
            ...
    </xpdl:WorkflowProcess>
```

Ainsi, ES fournit des constructions prédéfinies pour prendre en charge les contrôles de flux élémentaires. Le produit obtient une note de 2 sur 2.

6.3.2 Contrôle avancé du flux

jBPM

- Choix multiples : jPDL ne possède pas de construction prédéfinies pour réaliser une divergence à choix multiples. La fonction est possible avec du code Java. Une

implémentation est d'ailleurs fournie, à titre illustratif, avec la classe *ForEachForkAH*. L'implémentation des modèles BPMN permet de directement définir une divergence à choix multiples avec la balise *inclusiveGateway*.

- Convergence simple : Les implémentations jPDL et BPMN stockent les tâches provenant d'une convergence à choix multiples dans une collection de tâches. Il est aisé de faire que chaque tâche qui arrive au point de convergence continue automatiquement son exécution indépendamment des autres. Ceci réalise la convergence simple.
- Convergence synchronisante : La convergence synchronisante n'est pas implémentée malgré que le modèle standard BPMN la définit avec des portes *OR-JOIN*.
- Discriminateur : La notion de discriminateur n'est pas directement prise en charge, mais on peut lier la porte de convergence à une méthode Java qui poursuivrait l'exécution de la première branche qui l'atteint en ignorant les autres branches.

Ainsi, jBPM permet d'exprimer directement la notion de choix multiples avec son implémentation du modèle BPMN. Les autres contrôles de flux avancés sont réalisables avec des solutions de contournement. Si certains contrôles, tel que la convergence simple, sont faciles à coder ; d'autres, comme la convergence synchronisante et le discriminateur, nécessitent un certain effort. Le produit obtient une note de 1 sur 2.

Bonita Open Solution

- Choix multiples : Définir plusieurs transitions à partir d'un même élément et contraindre leur activation par des conditions non exclusives. Dans l'exemple de la figure 6.10, un client peut être à la fois entrepreneur, âgé de plus de 50 ans et avoir des enfants à charge. Les trois choix qui en découlent sont alors activés.
- Convergence multiple / synchronisante : Si on joint les branchements du choix multiple dans une porte de type *Join AND*, le processus est suspendu lorsqu'au moins un des choix n'a pas été activé, ce qui peut être le cas avec un choix multiple. Selon la spécification BPMN 2.0, une porte de type AND permettrait de réaliser une convergence multiple. Les portes AND ne sont actuellement pas prises en charge par Bonita.

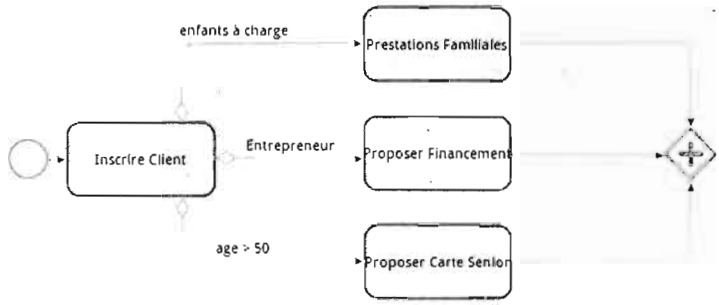


Figure 6.10 Choix multiples avec Bonita

Pour réaliser la convergence synchronisante, il faudrait définir les étapes du processus subséquentes à la divergence en choix multiples dans un sous-processus. Les branches du choix multiple se termineraient toutes par une transition vers une porte XOR qui amène au sous-processus. Une variable de processus *nbActivations* de type entier serait incrémentée dans chaque branche activée. Le sous-processus subséquent devra alors itérer *nbActivations* fois, réalisant ainsi le patron de convergence synchronisante. Pour obtenir la convergence multiple, il faudrait définir un connecteur dans chaque branche du choix multiple qui invoquerait le sous-processus dès qu'elle est activée.

- Discriminateur : L'implémentation Bonita de la porte XOR fonctionne comme un discriminateur : la première branche qui atteint la porte poursuit son exécution, les autres branches sont ignorées.

Ainsi, Bonita permet de représenter directement les choix multiples et les discriminateurs mais n'offre pas de portes AND pour réaliser la convergence multiple ou synchronisante. Ces fonctionnalités sont possibles avec des solutions de contournement (définir un sous-processus, des connecteurs, des variables de processus...). Le produit obtient une note de 1 sur 2 sur ce critère.

Enhydra Shark

- Choix multiples : ES implémente les choix multiples d'une façon élégante. Les transitions émanant de l'activité A dans la figure 6.11 portent une condition de type

<xpdl :Split Type="AND">. Une variable nommée *whereToGo* est définie comme attribut étendu de l'activité A. Si la valeur de la variable est B, C ou D, le processus poursuivra l'exécution à l'activité correspondante. Si *whereToGo* a une valeur AB, BA, ABC... alors toutes les activités mentionnées seront activées.

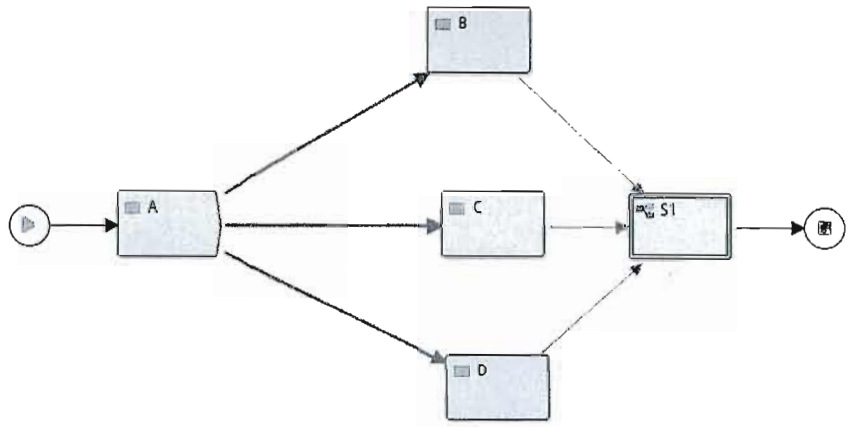


Figure 6.11 Choix et convergence multiples avec ES

- Convergence multiple : L'ensemble des activités subséquentes au choix multiple sont encapsulées dans un sous-processus asynchrone (type *SubFlow*. La convergence vers le sous-processus est conditionnée avec une jointure de type *XOR*.

```
<xpdl:Activity Id="S1">
    <xpdl:Implementation>
        <xpdl:SubFlow Id="WfPIMI_Subflow" Execution="ASYNCHR">
            <xpdl:ActualParameters>
                <xpdl:ActualParameter>common</xpdl:ActualParameter>
                <xpdl:ActualParameter>sbflw1data</xpdl:ActualParameter>
            </xpdl:ActualParameters>
        </xpdl:SubFlow>
    </xpdl:Implementation>
    <xpdl:TransitionRestrictions>
        <xpdl:TransitionRestriction>
            <xpdl:Join Type="XOR" />
        </xpdl:TransitionRestriction>
    </xpdl:TransitionRestrictions>
```

```
</xpdl:Activity>
```

- Convergence synchronisante : La variable *whereToGo* fonctionne selon le même principe que pour le choix multiple : Une valeur *B* déclenche l'activité *B*, une valeur *BC* déclenche les activités *B* et *C*, etc. La différence est que des routeurs (activités de type *Route*) s'intercalent entre les transitions. À la figure 6.12, le routeur *R1* transite vers l'activité *B* si la valeur de la variable *whereToGo* contient la lettre *B*, sinon l'activité est court-circuitée et la branche passant par *R1* transite directement vers *R3*. Le fonctionnement du routeur *R2* est similaire. La contrainte *Sinon* est représentée graphiquement par une flèche de couleur orange. Elle est exprimée en XPDL avec le type *OTHERWISE*.

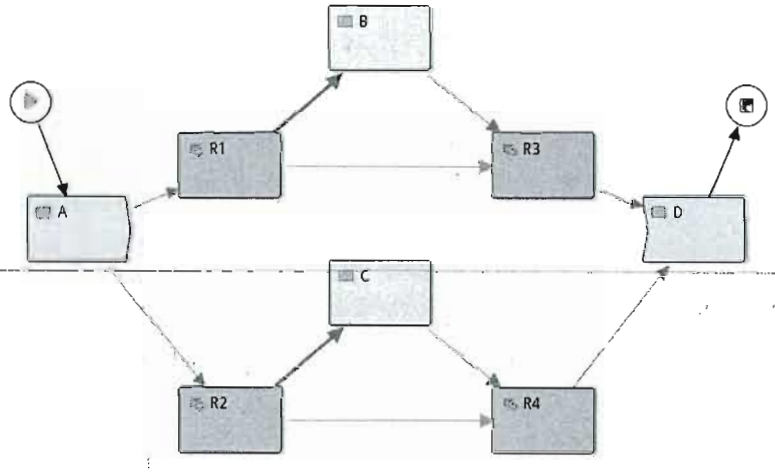


Figure 6.12 Choix multiples et convergence synchronisante avec ES

Le code suivant reprend les extraits XPDL les plus pertinents pour l'implémentation ES de la convergence synchronisante.

```
...
<xpdl:Activity Id="R1">
    <xpdl:Route/>
    ...
        <xpdl:TransitionRestriction>
            <xpdl:Split Type="XOR">
                <xpdl:TransitionRefs>
```

```
                <xpdl:TransitionRef Id="Tr1" />
                <xpdl:TransitionRef Id="Tr2" />
    ...
</xpdl:Activity>

<xpdl:Transition Id="Tr1" From="R1" To="B">
    <xpdl:Condition Type="CONDITION">whereToGo.indexOf('C') != -1</
        xpdl:Condition>
...
<xpdl:Transition Id="Tr2" From="R1" To="R3">
    <xpdl:Condition Type="OTHERWISE" />
...
```

- Discriminateur : Le discriminateur attend qu'une des branches entrantes complète son exécution avant d'enclencher les activités subséquentes. Si tel est le cas, le discrimateur attend la complétion de toutes les autres branches entrantes et les ignore. Une fois toutes les branches entrantes activées, le discriminateur est réinitialisé pour une prochaine utilisation. ES implémente le patron en définissant deux compteurs (variables entières) :
  - *openedBranchesAfterA* représente le nombre de branches activées dont les activités n'ont pas encore terminé leur exécution.
  - *availableBranchesToAccessE* représente le nombre de branches disponibles pour accéder à l'activité subséquente *E*.

  L'activité automatique *ResetDiscriminator* initialise le compteur *availablesBranches-ToAccessE* à 1 pour indiquer qu'une seule branche peut passer vers l'activité *E*. Quant au compteur *openedBranchesAfterA*, il est initialisé à zéro.

  Les activités automatiques avant les activités *B*, *C* et *D* incrémentent chacune la variable *openedBranchesAfterA* si la branche à laquelle elles appartiennent est activée. Les transitions émanant des activités *B*, *C* et *D* ont une contrainte de type divergence *XOR* (choix exclusif). Les transitions vers *EnsureEOnlyOnce* ne sont activées que si $availableBranchesToAccessE > 0$ (c'est à dire qu'aucune branche n'a encore atteint l'activité *E*). Sinon, l'activité transite vers *HandleRemaining*.

L'activité automatique *EnsureEOnlyOnce* décrémente les variables *openedBranchesAfterA* et *availableBranchesToAccessE*. Ainsi, la première activité déclenchée parmi *B*, *C* et *D* qui termine son exécution est la seule à transiter vers *EnsureEOnlyOnce*. Le compteur *availableBranchesToAccessE*, initialisé à 1 est alors décrémenté. Lorsque les autres activités déclenchées selon le mécanisme de choix multiples terminent leur exécution, elles transitent vers l'activité automatique *HandleRemainingBranches* qui décrémente *openedBranchesAfterA*.

Enfin, les transitions de *HandleRemaining* vers la fin et de *E* vers la fin ne sont activées que si la valeur du compteur *openedBranchesAfterA* $= 0$, c'est à dire que toutes les branches alternatives déclenchées ont terminé leur exécution.

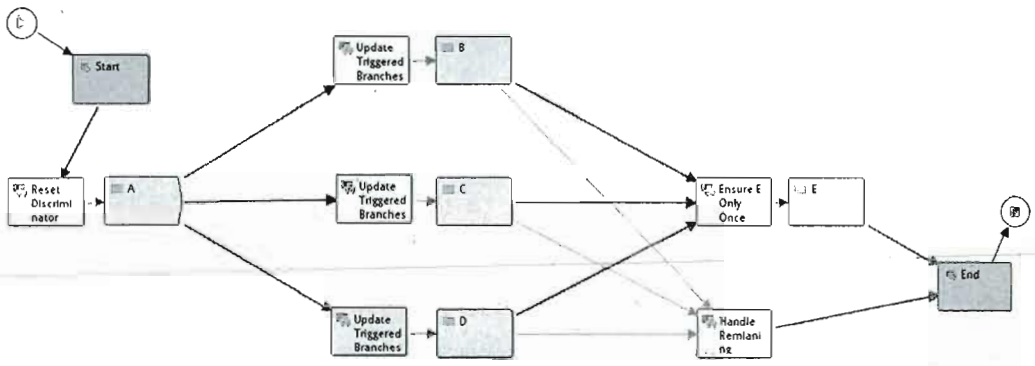


Figure 6.13 Discriminateur avec ES

ES propose des implémentations des patrons de contrôle de flux avancés. Il obtient la note de 2 sur 2.

6.3.3 Contrôle selon l'état du processus

jBPM

- Choix différés : jBPM permet, pour les deux langages pris en charge, de lier le choix de la branche à sélectionner dans un point de divergence à une valeur connue pendant l'exécution. Ceci est réalisable en liant la condition de sélection à une variable de

processus comme le montre l'exemple jPDL suivant :

```
<decision name="evaluer document" expr="#{contenu}" >
  <transition name="acceptable" to="soumettre document" />
  <transition name="insuffisant" to="refuser document"  />
```

- Routage parallèle interfolié : La définition d'un ordre partiel n'est pas directement exprimable avec jBPM. Il est cependant possible de réaliser le patron en plaçant les étapes qui doivent respecter un ordre d'exécution dans une même branche et les autres étapes dans des branches parallèles. Via du code Java, on établirait un mécanisme de verrou pour s'assurer qu'une seule étape à la fois peut être en cours d'exécution.
- Arrivée à un stade (milestone) : La notion de *milestone* n'est pas exprimable avec jBPM. Toutefois, on peut placer un état d'attente avant l'activité conditionnée par l'arrivée à un stade. Lorsque le stade qui se trouve dans une branche parallèle est atteint, une activité automatique signale la poursuite du flux à l'état d'attente et démarre ainsi l'activité subséquente.

jBPM permet d'exprimer directement les choix différés mais le routage interfolié et le concept de *milestone* doivent être implémentés avec des méthodes Java. Le produit obtient une note de 1 sur 2.

Bonita Open Solution

- Choix différés : Les conditions d'activation de chaque branche sont exprimées selon des variables dont les valeurs ne sont connues qu'à l'exécution.
- Routage parallèle interfolié : Les sous-ensembles de tâches ordonnées sont placés dans une branche unique d'un ensemble de branches parallèles, forçant ainsi leur exécution selon une séquence. Les différentes branches parallèles convergeraient vers une porte AND. Pour faire en sorte que deux tâches dans deux branches parallèles ne puissent s'exécuter en même temps, on pourrait implémenter un mécanisme de verrou via une variable de processus. Le code de lecture et de modification du verrou serait implémenté dans des connecteurs Java (ou Groovy) à l'entrée et à la sortie de chaque étape de l'ordre partiel.

- Arrivée à un stade (milestone) : Une étape *milestone* peut déclencher l'exécution d'une autre étape du processus en utilisant un connecteur Bonita de type *Execute a Step* (exécuter une étape). Le connecteur prend en paramètre le nom de l'étape à exécuter. Cette dernière doit posséder un connecteur, déclenché à l'entrée de l'étape. Le connecteur a pour rôle de suspendre la tâche.

Ainsi, les patrons de choix différés et d'arrivée à un stade sont directement exprimables avec Bonita. Le routage parallèle interfolié nécessite un certain effort de développement. Le produit obtient la note de 1 sur 2.

Enhydra Shark

- Choix différé : Les transitions vers les branches alternatives sont conditionnées par des expressions dont les valeurs ne sont connues qu'à l'exécution.
- Routage parallèle interfolié : De manière similaire aux deux autres produits, on pourrait mettre les activités ayant un ordre partiel dans des branches à part. À ce titre, ES fournit les notions d'*Activity Set* et *Block Activity*. Un *Activity Set* est un ensemble indépendant d'activités. Un *Block Activity* représente un *Activity Set* dans le processus parent. Afin d'assurer l'exécution d'un seul bloc d'activités à un instant donné, il est nécessaire d'implémenter un mécanisme d'exclusion mutuelle, par exemple avec une variable partagée faisant office de verrou.
- Arrivée à un stade : En manipulant l'API d'ES ou en utilisant des agents, on suspend l'activité dont l'exécution dépend de l'arrivée à un stade. Par ailleurs, on définit une activité automatique qui s'exécute à l'arrivée au stade et qui reprend l'activité suspendue, en utilisant la méthode *void resumeActivity(WMSessionHandle shandle, String identifiantProcessus, String identifiantActivite)* de l'interface *ExecutionAdministration*.

ES n'implémente pas directement les trois patrons de contrôle selon l'état du processus. Ils sont toutefois réalisables avec des solutions de contournement. Le produit obtient une note 1 sur 2.

6.3.4 Itération dans le flux

jBPM

- Boucles : jBPM ne fournit pas d'opérateur de bouclage pour répéter l'exécution d'éléments du flux. Même si la norme BPMN permet de boucler sur une activité selon une construction *pour..faire* ou *tant que..faire*, l'implémentation du standard par jBPM ne fournit pas une manière directe de l'exprimer.
- Cycles arbitraires : Les transitions entre éléments du flux peuvent être définies de manière à former des cycles dont le nombre de répétitions serait lié à la valeur d'une variable.
- Récursivité : Une tâche jBPM pourrait créer de nouvelles instances du processus courant. Cependant, il n'existe pas de manière directe de signifier au processus parent d'attendre la fin de l'exécution des instances créées pour poursuivre. La récursivité n'est donc pas directement prise en charge.

Les cycles arbitraires sont directement réalisables avec jBPM. Le produit ne fournit pas d'opérateur de bouclage et ne permet pas de définir des processus récursifs à l'aide d'instructions prédéfinies. La note accordée est de 1 sur 2.

Bonita Open Solution

- Boucles : Bonita prend directement en charge le bouclage. Il suffit de cocher le bouton radio *is Looped* dans les propriétés avancées de l'activité à boucler. La condition de bouclage peut être exprimée sous forme d'expression Groovy complexe, comprenant des variables et des méthodes. La vérification de la condition se fait au début ou à la fin de l'exécution de l'étape (comportement d'une boucle *for* ou d'une boucle *do..while*).
- Cycles arbitraires : Les transitions entre éléments du flux peuvent être définies de manière à former des cycles dont le nombre de répétitions serait lié à la valeur d'une variable.
- Récursivité : Un connecteur Java ou Groovy est défini à l'étape où une nouvelle

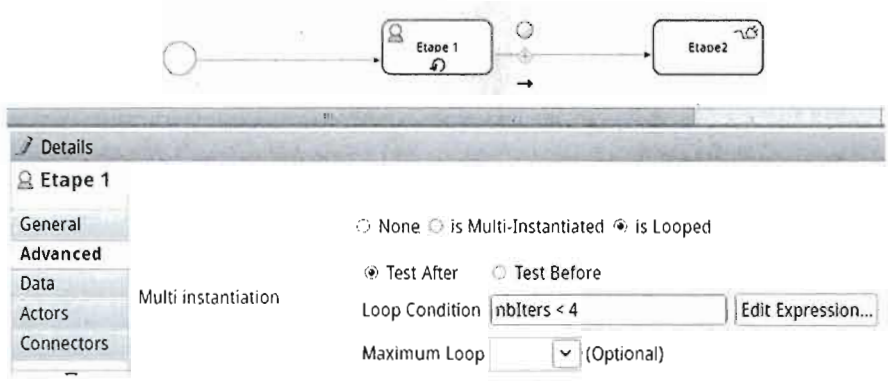


Figure 6.14 Boucles avec Bonita

instance du processus sera créée. Le connecteur utilise l'API de Bonita comme suit :

```
Map<String, Object> vars = new HashMap<String, Object>();
// Recuperer les valeurs de toutes les variables
// de l'instance de processus et de ses etapes
Map<Object> variablesDeProcessus = AccessorUtil.getQueryRuntimeAPI().
    getProcessInstanceVariables(processInstanceUUID);
vars.putAll(variablesDeProcessus);
Map<Object> variablesDuneActivite = AccessorUtil.getQueryRuntimeAPI().
    getActivityInstanceVariables(activityUUID)
vars.putAll(variablesDuneActivite);
...
// creer la nouvelle instance en lui passant
// les valeurs des variables comme parametres
AccessorUtil.getRuntimeAPI().instantiateProcess(new
    ProcessDefinitionUUID("NomDuProcessus--Version"), vars);
```

Ainsi, les boucles et les cycles arbitraires sont directement pris en charge. La récursivité est réalisable en passant par l'API Bonita, sans trop d'efforts de programmation. Le produit obtient la note de 2 sur 2.

Enhydra Shark

- Cycles arbitraires : Établir une transition vers une étape antérieure du processus et une autre vers une étape subséquente. Les conditions d'évaluation des deux transitions sont mutuellement exclusives. Elles sont exprimées en fonction de variables dont les valeurs peuvent changer plusieurs fois pendant l'exécution, de sorte à avoir un cycle d'exécution à chaque fois que la première transition est activée.
- Boucles : ES ne possède pas de constructions prédéfinies pour exprimer les boucles. Une activité *A* boucle sur elle-même en créant une transition de *A* vers elle-même et une deuxième transition de *A* vers les activités suivantes . Le mécanisme de cycles arbitraires est alors utilisé pour déterminer le nombre d'itérations à l'exécution.
- Récursivité : Le flux de travail s'invoque lui même par l'intermédiaire d'un élément de type *SubFlow* qui référence le processus appelant.

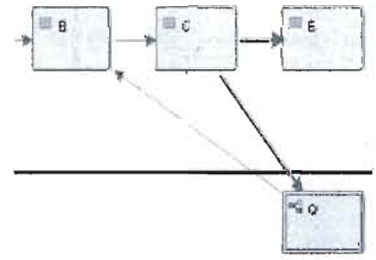


Figure 6.15 Récursivité avec ES

Les extraits suivants montrent les éléments XPDL pertinents de la solution.

```
<xpdl:WorkflowProcess Id="Recursivite">
  ...
  <xpdl:Activity Id="Q">
    <xpdl:Implementation>
      <xpdl:SubFlow Id="Recursivite"/>
    </xpdl:Implementation>
  </xpdl:Activity>
  ...
</xpdl:WorkflowProcess>
```

ES implémente donc les cycles arbitraires et la récursivité mais ne propose pas de construction prédéfinie pour définir des boucles. Le produit obtient une note de 1 sur 2 pour ce critère.

6.3.5 Annulation d'éléments dans le flux

jBPM

- Annuler une tâche : L'interface *TaskService* définit les méthodes suivantes pour annuler une tâche avant de l'avoir complétée :

```
deleteTask(String idTache)
deleteTask(String taskId, String raison)
deleteTaskCascade(String idTache)
```

  La seconde méthode permet d'indiquer la raison de l'annulation et la troisième méthode annule la tâche ainsi que ses informations journalisées.
- Annuler un processus : L'interface *ExecutionService* définit les méthodes suivantes pour annuler une instance de processus :

```
deleteProcessInstance(String idInstanceProcessus)
deleteProcessInstanceCascade(String idInstanceProcessus)
```

  La seconde méthode supprime les informations journalisées relatives à l'instance de processus.
- Annuler une région : L'ensemble des activités formant une région est groupé dans un sous-processus. Les méthodes d'annulation de processus peuvent y être appliquées. Pour pouvoir former un sous-processus, les activités doivent être interconnectées.

Ainsi jBPM permet d'annuler une tâche et une instance de processus. L'utilisateur choisit de garder ou de supprimer de l'historique d'exécution les informations relatives à la tâche ou à l'instance de processus annulée. Étant donné que l'annulation de région n'est possible que lorsque les activités membres de la région forment un sous-processus, on attribue une note de 1 sur 2 sur ce critère.

- Annuler une tâche : Un événement intermédiaire de fin (*intermediate end event*) est attaché à l'activité pouvant être annulée. La transition de l'activité à l'événement de fin est conditionnée par une expression dont la valeur est fausse par défaut. Lors de l'exécution, l'expression est mise à vrai si on désire annuler la tâche.

  Par ailleurs, les événements de types *throw link* et *catch link* peuvent remplir le même rôle que l'événement intermédiaire de fin : L'activité à annuler transite vers un *throw link*. Le *catch link* qui lui est associé intercepte le flux et démarre la séquence d'activités à réaliser lorsque la tâche en cours est annulée.
- Annuler un processus : La classe *RuntimeAPIBean* de l'API de Bonita fournit les méthodes suivantes pour annuler une instance de processus ou une collection d'instances de processus :

```
cancelProcessInstance(identifiantInstanceProcessus)}
cancelProcessInstances(Collection$<$identifiantInstanceProcessus$>$)
```

  L'interface web User XP encapsule ces fonctions en permettant à l'administrateur :
  - de sélectionner dans une liste une ou plusieurs instances de processus actives à annuler.

Figure 6.16 Annulation d'un ensemble d'instances de processus actives

  - de sélectionner dans une liste une ou plusieurs définitions de processus dont toutes les instances seront supprimées.
- Annuler une région : Regrouper les processus de la région à annuler dans un sous-processus et appliquer la procédure d'annulation de processus. Toutefois, cette ap-

proche n'est valable que si la région est constituée d'éléments interconnectés dans le processus original.

L'annulation d'instance de processus est directement prise en charge par Bonita, ce qui n'est pas le cas de l'annulation de tâches et de régions de tâches. Le produit obtient une note de 1 sur 2.

Enhydra Shark

- Annuler une tâche : L'interface *WfExecutionObject*, implémentée par la classe *WfActivity*, expose une méthode *void abort()* qui requiert l'abandon d'un objet d'exécution suspendu avant qu'il ait terminé son exécution. L'état de l'objet est mis à *closed.aborted*. Il est donc possible de suspendre puis annuler l'exécution d'une tâche.
- Annuler un processus : La méthode suivante permet d'annuler l'exécution d'une instance de processus passée en paramètre.

```
void abortProcessInstance(WMSessionHandle shandle, String
    idInstanceProcessus)
```

La méthode ci-dessous permet d'annuler un groupe d'instances de processus créées à partir d'une définition de processus passée en paramètre. Un filtre optionnel permet de cibler un sous-ensemble des instances du processus.

```
void abortProcessInstances(WMSessionHandle shandle, String
    idDefinitionProcessus, WMFilter filtre)
```

- Annuler une région : Regrouper les processus de la région à annuler dans un sous-processus et appliquer la procédure d'annulation de processus. Les tâches de la région doivent toutefois être interconnectées, de sorte à former un sous-processus.

Les fonctionnalités d'annulation de tâches et d'instances de processus sont directement accessibles à partir des interfaces graphiques d'administration Swing et web (*sharkWebClient*).

L'API d'ES expose des méthodes d'annulation de tâches et d'instances de processus. Ces fonctionnalités sont accessibles dans les applications d'administration du produit.

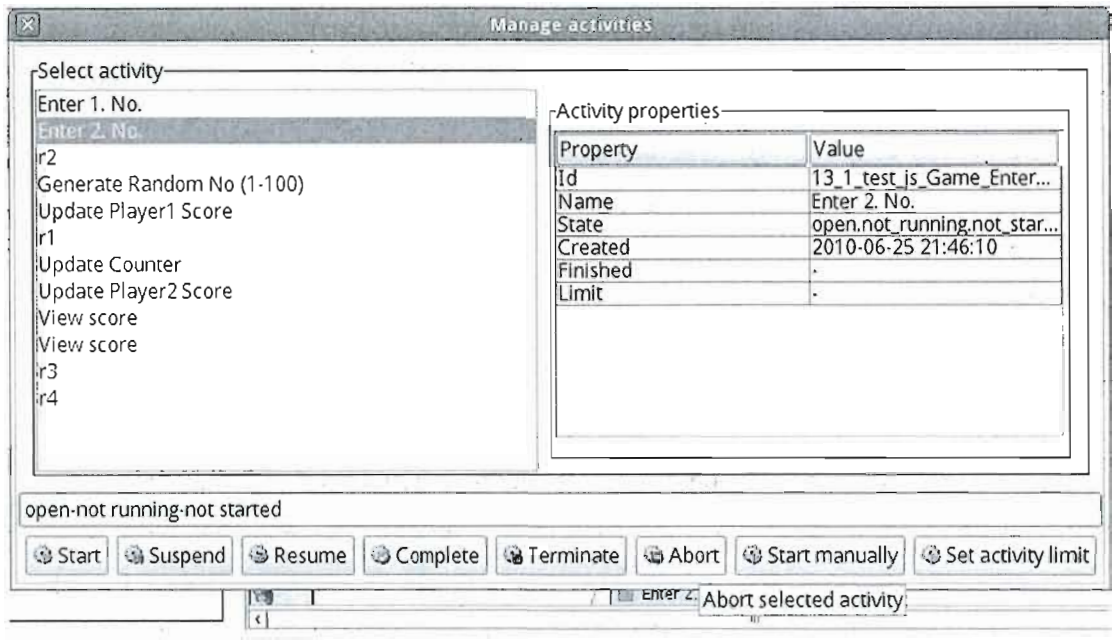


Figure 6.17 Annulation d'une tâche à partir de l'interface graphique ES

Étant donné que l'annulation de régions n'est possible que lorsque les activités membres de la région forment un sous-processus, on attribue une note de 1 sur 2 sur ce critère.

6.3.6 Gestion d'instances multiples

jBPM

Une tâche jBPM peut créer de nouvelles tâches. Le nombre d'instances de tâches peut être défini à la conception, en spécifiant une valeur littérale dans le code. Il peut également être connu à l'exécution, en définissant une variable qui serait initialisée au démarrage du processus.

Ainsi, les patrons de création d'instances multiples sont indirectement réalisables avec jBPM. Le produit obtient une note de 1 sur 2, car un certain effort de développement est nécessaire.

La création de plusieurs instances d'une tâche nécessite un connecteur de type instanciateur (*Instantiator*) et un autre de type contrôleur de jointure (*Join Checker*). L'instanciateur peut être configuré pour créer un nombre fixe d'instances de l'étape ou un nombre variable connu à l'exécution ou encore utiliser un script Groovy ou une classe qui étend *MultiInstantiator* de l'API de Bonita. L'instanciateur permet d'affecter des valeurs aux données requises par chaque instance de l'étape multi-instanciée, lors de son initialisation.

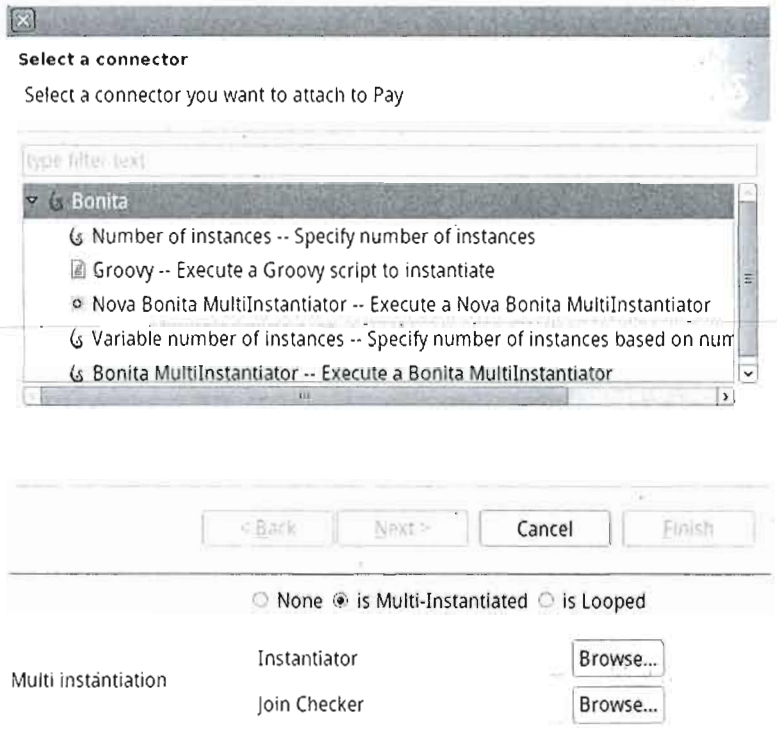


Figure 6.18 Configuration de l'instanciateur pour une étape à instances multiples

Le contrôleur de jointure permet de définir les conditions de poursuite de l'exécution du flux à partir des instances multiples parallèles (Par exemple, le flux est poursuivi si 70% des instances multiples ont terminé leur exécution).

Enfin, il est possible de définir une étape de type sous-processus comme multi-instanciée

et créer ainsi plusieurs instances parallèles d'un ensemble de tâches.
Bonita permet directement la gestion d'instances multiples de tâches et obtient une note de 2 sur 2.

Enhydra Shark

ES fournit des implémentations des patrons d'instances multiples connues à la conception avec et sans synchronisation. Les solutions proposées définissent plusieurs activités de types *SubFlow* qui se réfèrent à un même flux de travail. La transition de l'activité *A* vers les instances multiples se fait selon une divergence (*<xpdl :Slip Type="AND">*). Les instances multiples peuvent partager des données communes et posséder des données spécifiques. La différence entre les solutions synchrones et asynchrones réside dans l'attribut *Execution* des éléments *SubFlow* (défini à *SYNCHR* ou *ASYNCHR*).
Instance synchrone :

```
<xpdl:SubFlow Execution="ASYNCHR" Id="Instance1">
```

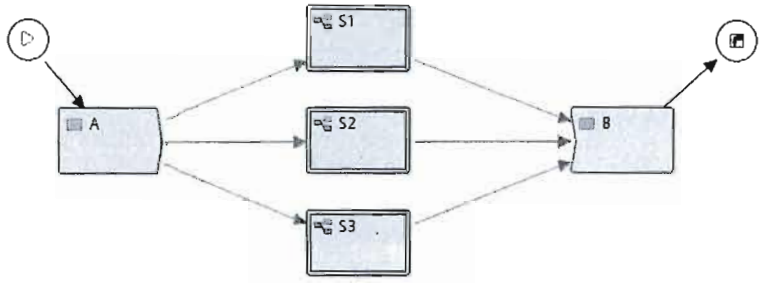


Figure 6.19 Instances multiples connues à la conception avec ES

Il est possible de créer un nombre d'instances multiples connu à l'exécution, en définissant une boucle où un élément *SubFlow* est invoqué à chaque itération. Toutefois, ces instances multiples ne sont pas synchronisables car elles ne doivent pas bloquer le déclenchement en parallèle de plusieurs instances.
Ainsi, ES permet la création d'instances multiples dont le nombre est connu à la conception. La création d'un nombre arbitraire d'instances nécessite une solution de contournement. Le produit obtient la note de 1 sur 2 pour ce critère.

6.3.7 Démarrage déclenché par des signaux externes

jBPM

- Déclencheur éphémère : Le passage d'un état (*state*) jPDL à l'état suivant se fait lorsque l'état en attente reçoit un signal externe.

```
ProcessInstance instanceProcessus = executionService.
    startProcessInstanceByKey("processusDemandeEmprunt");
Execution execution = instanceProcessus.findActiveExecutionIn("
    saisirDemande");
instanceProcessus = executionService.signalExecutionById(execution.getId
    ());
```

L'exécution des tâches BPMN peut être déclenchée par des signaux externes :

```
ProcessInstance instanceProcessus = executionService.
    startProcessInstanceByKey("processusValidationClient");
executionService.signalExecutionById(instanceProcessus.
    findActiveExecutionIn("tacheValiderNom").getId());
```

Enfin, un *timer* jPDL est un chronomètre qui se déclenche suite à un événement externe

```
<state name="etatA" >
    <on event="evenementDeclencheur">
      <timer duedate="10 minutes"/>
      <event-listener class="ClasseImplementantEventListener" />
    </on>
    <transition to="prochaine etape" />
</state>
```

- Déclencheur persistant : Les déclencheurs persistants ne sont pas pris en charge par jBPM.

Ainsi, jBPM gère directement les déclencheurs éphémères mais pas les déclencheurs persistants. Le produit obtient une note de 1 sur 2 pour ce critère.

Au sein d'un même processus, les événements de type *Throw Link* sont interceptés par des événements de type *Catch Link*. Ainsi, un lien de type *throw* constitue un déclencheur pour le démarrage de la séquence de flux vers laquelle transite le lien de type *catch*.

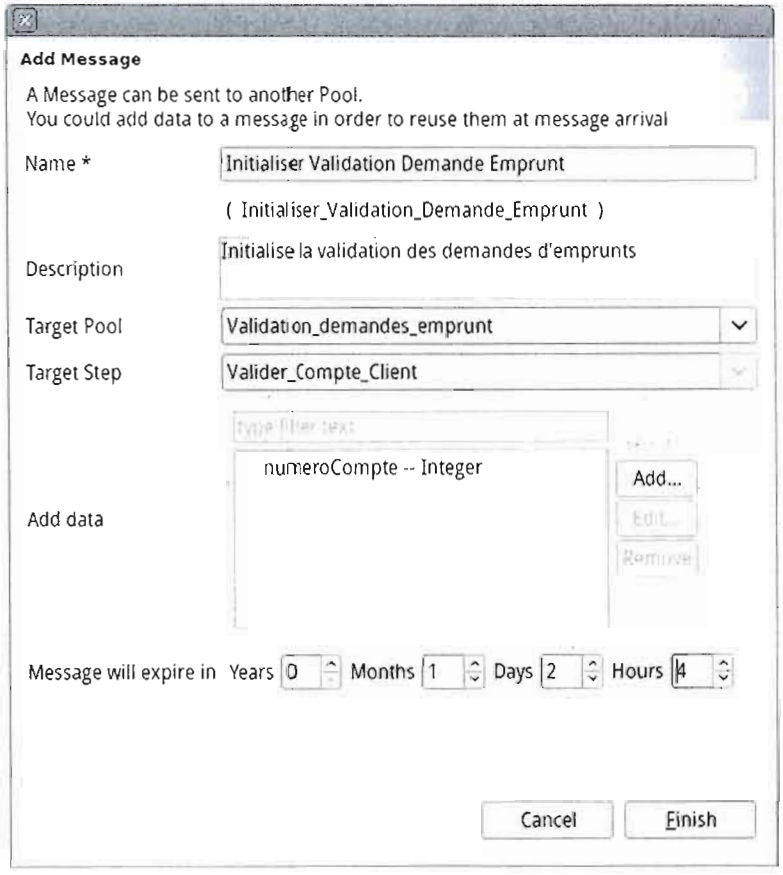


Figure 6.20 Démarrage déclenché par un message de type throw

Les messages de types *Throw* et *Catch* sont équivalents aux liens, à la différence qu'ils circulent entre plusieurs processus définis dans des bassins (*pools*) différents. La figure 6.20 montre la configuration d'un message qui va déclencher le processus *Validation_demandes_emprunt* à l'étape *Valider_Compte_Client* en lui passant comme pa-

ramètre le numéro du compte client. Une propriété permet de définir le délai d'expiration des messages, ce qui en fait d'éventuels déclencheurs externes persistants.
La partie *throw* des liens et des messages peut survenir suite à l'invocation de toute sorte de systèmes externes via des connecteurs. Par exemple, un connecteur pourrait invoquer un service web et, selon la valeur retournée, transiter vers un message ou un lien *throw*.
Ainsi, Bonita permet de démarrer l'exécution de tâches par des déclencheurs externes éphémères ou persistants (selon leur délai d'expiration). Le produit obtient une note de 2 sur 2.

Enhydra Shark

- Déclencheur éphémère : L'attribut de date limite (*deadline*) des activités permet de définir des déclencheurs éphémères. Les dates limites peuvent être spécifiées pour des activités de type *Route* qui constitueront des états d'attente. Les agents peuvent être utilisés pour réaliser des déclencheurs éphémères. Par exemple, un agent de réception de courriel déclencherait la poursuite du flux de travail à la réception de l'événement externe *arrivée d'un courriel*.
- Déclencheur persistant : Afin de recevoir un déclencheur, le flux d'exécution doit se trouver à l'activité en attente du signal, ce qui correspond à la notion de déclencheur éphémère. ES ne prend donc pas en charge les déclencheurs persistants.

Une note de 1 sur 2 est accordée pour ce critère, qui n'est satisfait qu'à moitié.

6.4 Distribution des items de travail

6.4.1 Distribution à la conception

jBPM

L'interface *IdentityService* de jBPM fournit les méthodes de gestion des utilisateurs :
- La création des utilisateurs se fait avec les méthodes :

```
createUser(String idUtilisateur, String prenom, String nom)
createUser(String idUtilisateur, String prenom, String nom, String
    courriel)
```

- La création des groupes est réalisée par :

```
createGroup(String nomGroupe)
createGroup(String nomGroupe, String typeGroupe)
createGroup(String nomGroupe, String typeGroupe, String idGroupeParent)
```

Les trois variantes retournent l'identifiant généré du groupe.

- Pour qu'un utilisateur soit membre d'un groupe, on utilise :

```
createMembership(String idUtilisateur, String idGroupe)
createMembership(String idUtilisateur, String idGroupe, String role)
```

La seconde méthode permet en plus d'affecter un rôle à l'utilisateur.

La distribution des tâches à la conception est évaluée pour cinq variantes :

- Distribution directe : L'attribut *assignee* permet de spécifier à la conception l'identité de l'utilisateur qui effectuera une tâche.

```
identityService.createUser("ZGHO1", "Zeineb", "Ghozlani");
<task name="nom tache"
    assignee="ZGHO1">
    <transition to="prochaine etape" />
</task>
```

Pour proposer une tâche à un groupe, on utilise l'attribut *candidate-groups* :

```
<task name="nom tache"
        candidate-groups="identifiant groupe">
```

Il est possible de réaliser l'attribution de tâches dans une classe qui implémente l'interface *AssignmentHandler*.

Enfin, jPDL permet d'affecter un ensemble de tâches, regroupées dans un couloir (*swimlane*), à un utilisateur ou un groupe.

```
<swimlane name="taches de validation"
            candidate-groups="verificateurs" />
...
```

```
<task name="valider nom client"
        swimlane="taches de validation">
</task>

<task
      name="valider balance compte"
      swimlane="taches de validation">
</task>
```

L'implémentation jBPM de la notation BPMN reprend les mécanismes utilisés pour le langage jPDL. Par exemple, le code suivant permet de proposer une tâche à un groupe :

```
<userTask id="idTache" name="nom tache">
  <potentialOwner resourceRef="verificateurs" jbpm:type="group">
    <resourceAssignmentExpression>
      <formalExpression>verificateurs</formalExpression>
    </resourceAssignmentExpression>
  </potentialOwner>
</userTask>
```

- Distribution basée sur le rôle : Les méthodes de gestion des utilisateurs citées précédemment permettent d'affecter un rôle à une ressource. L'implémentation actuelle de BPMN 2.0 par jBPM permet d'allouer des tâches avec une expression d'allocation *resourceAssignmentExpression*, combinée à l'identifiant d'un ensemble d'acteurs potentiels (*PotentialOwner*).

  Dans l'exemple suivante, la tâche *uneTache* est proposée à tous les participants ayant le rôle *expertsComptables*.

```
<userTask id="uneTache" name="Une tache">
  <potentialOwner resourceRef="expertsComptables" jbpm:type="role">
    <resourceAssignmentExpression>
      <formalExpression>expertsComptables</formalExpression>
    </resourceAssignmentExpression>
  </potentialOwner>
</userTask>
```

Par ailleurs, le module de gestion d'identité par défaut peut être remplacé par une

implémentation personnelle qui gèrerait le concept de rôle.

- Distribution basée sur l'instance actuelle : Les tâches d'un processus peuvent être associées à un couloir. L'instance de processus retient l'identité des candidats et de l'utilisateur qui a effectué la première tâche dans le couloir. Les tâches suivantes du couloir seront assignées à ces candidats ou à cet utilisateur.
- Distribution différée : Les tâches sont assignées à des variables dont la valeur n'est connue qu'à l'exécution.
- Exécution automatique : jBPM permet de définir des tâches automatiques codées en Java, avec un langage de script ou encore avec les langages de sélection SQL et HQL.

Ainsi, jBPM est capable d'exprimer la distribution directe, différée, basée sur l'instance actuelle ou sur le rôle, ainsi que l'exécution automatique de tâches. Le produit obtient une note de 2 sur 2.

Bonita Open Solution

Les comptes d'utilisateurs et leurs rôles sont définis par l'administrateur à partir de l'application web User XP. Dans la figure 6.21, on crée un utilisateur et on lui attribue le rôle *Utilisateurs*

Les rôles sont gérés de la même façon, en sélectionnant l'onglet *Roles.*

Une fois les utilisateurs et les rôles définis, on alloue ou on propose chaque tâche non automatique à un utilisateur ou à un groupe d'utilisateurs. Il est possible d'affecter un utilisateur ou un groupe d'utilisateurs à toutes les tâches non automatiques d'un couloir entier (*lane*). La répartition des tâches sur les participants est alors visible dans la représentation graphique des processus. Bonita propose plusieurs types de groupes d'utilisateurs : l'initiateur du processus, une liste de noms d'utilisateurs, le rôle des utilisateurs, une liste de noms récupérés à partir d'un dépôt LDAP ou encore un rôle Talend MDM (Master Data Management)[13].

- Distribution directe : Chaque étape non automatique d'un processus possède un onglet

13. Talend est un outil d'intégration de données à code ouvert

Figure 6.21 Gestion des utilisateurs dans User XP

Actors permettant de désigner la ressource en charge de l'exécution. Si la ressource est individuelle, son nom est indiqué.

S'il s'agit d'un groupe d'utilisateurs, on pourrait :

- sélectionner le type *User List* et indiquer les noms des utilisateurs
- récupérer les noms à partir d'un dépôt LDAP ou de l'application Talend.

Dans les trois cas, les noms des utilisateurs sont connus à la conception.

- Distribution basée sur les rôles : Le type de groupe d'utilisateurs *User Role* prédéfini par Bonita sert à indiquer le rôle que doivent avoir les ressources pour exécuter une étape. Les rôles sont définis et attribués aux utilisateurs via l'interface UserXP.
- Distribution différée : Le type de groupes d'utilisateurs *User List* permet d'utiliser des variables de processus et des expressions Groovy pour les noms des utilisateurs. Leurs valeurs sont résolues pendant l'exécution.

  Par ailleurs, le type *Process Initiator* attribue la tâche à la ressource qui a initié le processus. Cette ressource n'est connue qu'au moment de l'exécution.

- Distribution basée sur l'instance actuelle : Le type *Process Initiator* permet d'attribuer un ensemble de tâches à l'initiateur de l'instance de processus, déterminé lors de la création de l'instance.
- Exécution automatique : Les tâches non humaines sont exécutées automatiquement par le système, sans avoir à les assigner à une ressource.

Bonita permet de facilement et directement distribuer les tâches aux ressources à la conception, à l'exécution, selon les rôles et selon l'instance actuelle. Les tâches non humaines sont automatiquement exécutées par le système. Le produit obtient une note de 2 sur 2.

Enhydra Shark

- Distribution directe : Les applications d'administration du système gèrent les utilisateurs d'ES. Un utilisateur est authentifié par son identifiant et son mot de passe. Des informations tel que son adresse courriel sont optionnels. Pour chaque processus à instancier, on établit une correspondance directe entre les participants définis dans le processus et les utilisateurs du système. À l'exécution, les activités attribuées à un participant sont affectées à l'utilisateur qui lui correspond.
- Distribution basée sur le rôle : La correspondance directe peut se faire entre les participants définis dans le processus et des groupes d'utilisateurs. Les applications d'administration du produit permettent d'ajouter, supprimer et modifier les groupes. À l'exécution, une activité attribuée à un participant sera proposée à tous les membres du groupe d'utilisateurs qui lui correspond.

  De plus, il est possible de se connecter à des répertoires LDAP et d'en importer des utilisateurs et des groupes pour les utiliser comme participants aux flux de travail.
- Distribution différée : Les tâches d'un processus sont associées à participant *varP* dont l'identité est connue à l'exécution.
- Distribution basée sur l'instance actuelle : La classe *HistoryRelatedAssignmentManager* est un gestionnaire d'offre d'activités qui utilise les attributs étendus XPDL. Une activité ayant l'attribut *ReassignToOriginalPerformer* assigne une activité exécutée

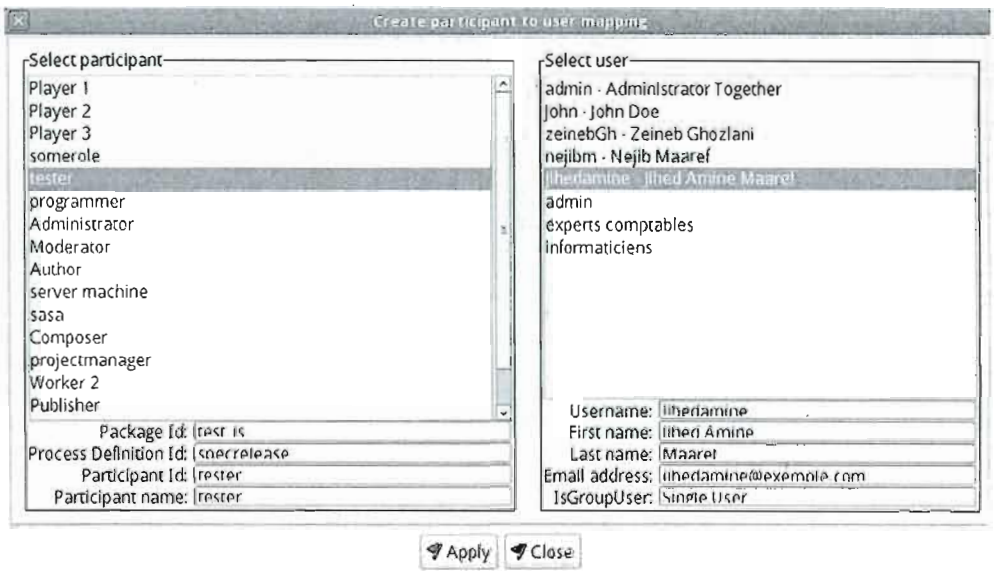


Figure 6.22 Correspondance entre participants et utilisateurs

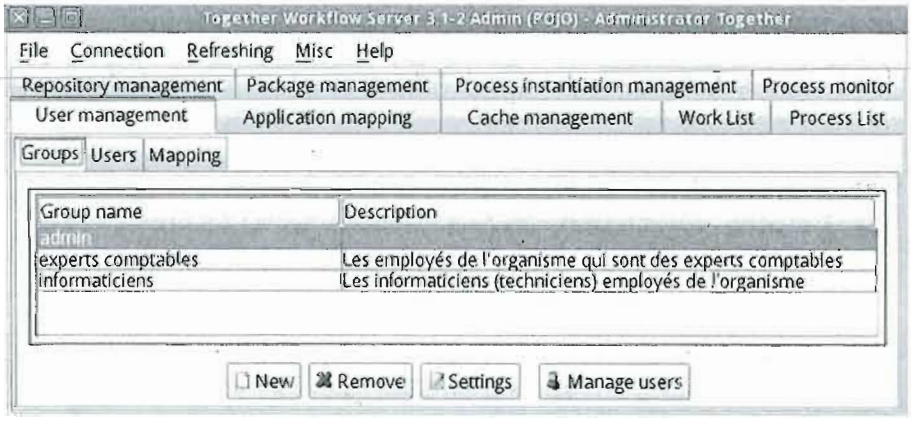


Figure 6.23 Gestion des groupes avec ES

plusieurs fois dans une même instance de processus à l'utilisateur ayant exécuté la première instance. L'attribut *AssignToPerformerOfActivity* force l'offre d'une activité à l'utilisateur ayant exécuté l'activité précédente. Enfin, l'attribut *DoNotAssignTo-PerformerOfActivity* interdit d'offrir l'activité à l'utilisateur ayant exécuté l'activité précédente. Un seul des trois attributs peut être associé à une activité donnée.

- Exécution automatique : Définir des tâches qui sont automatiquement démarrées par des agents. Les applications d'administration réalisent la correspondance entre les tâches automatiques définies dans les processus et les agents en charge de leur exécution.

Ainsi, ES permet la distribution directe, différée, basée sur le rôle ou sur l'instance actuelle. L'exécution automatique des tâches est réalisée par les agents. Le produit obtient une note de 2 sur 2.

6.4.2 Allocation des ressources par le système

jBPM

jBPM est capable à la fois d'offrir et d'assigner des tâches à un utilisateur ou à un groupe d'utilisateurs.

- Offrir une tâche : Une tâche peut être offerte à un ensemble d'un à plusieurs utilisateurs (attribut *candidate-users*) ou à un ensemble d'un à plusieurs groupes (attribut *candidate-groups*) :

```
<task name="nom tache"
      candidate-users="idUtilisateur1 [, idUtilisateur2]">
</task>

<task name="nom tache"
      candidate-groups="idGroupe1 [, idGroupe2]">
</task>
```

- Assigner une tâche : Les tâches sont assignées à un utilisateur dont on saisit l'identifiant.

```
<task name="nom tache"
        assignee="idUtilisateur">
</task>
```

- Stratégies de sélection : jBPM ne permet pas directement de sélectionner une ressource à laquelle une tâche est assignée, selon une stratégie de sélection prédéfinie.

En délégant l'assignation à une classe qui implémente l'interface *AssignmentHandler*, on est capable d'implémenter une stratégie de sélection. Le codage des stratégies de sélection demande toutefois un effort de développement important.

```
public NomDeLaClasse implements AssignmentHandler {

  public void assign(Assignable assignable, OpenExecution execution) {
        //Code pour selectionner une ressource a laquelle on assigne la
            tache
        //selon une strategie donnee (round robin, acteur ayant la file
        //la plus courte...)
        ...
        //L'assignation se fait avec la methode setAssignee
        assignable.setAssignee(ressourceChoisie);
  }
}
```

- Moment de distribution : Par défaut, la ressource à laquelle une tâche est offerte/assignée est désignée à la conception. La distribution se fait lors de l'activation du processus.

Le système peut, à sa propre initiative, offrir ou assigner des tâches aux ressources. Il n'est cependant pas possible de sélectionner les ressources selon une stratégie ou encore de réaliser une distribution précoce ou tardive des tâches avec des constructions des langages jPDL ou BPMN. Le produit n'obtient que 1 sur 2 pour ce critère.

Bonita Open Solution

- Offrir ou assigner le travail : Lorsque la tâche est associée à une ressource unique, elle lui est offerte et ne figure que dans sa boîte de réception UserXP.

  Lorsque la tâche est associée à un groupe, elle figure dans les boîtes de réception UserXP de tous les membres du groupe. Si un utilisateur la réclame (en sélectionnant la tâche et en cliquant sur *assign to me* dans l'interface User XP), elle est retirée des autres boîtes de réception.

  Dans les deux cas, Bonita offre la tâche à la/les ressource(s), en la déposant dans leurs

boîtes de réception. La décision d'exécuter ou non la tâche revient à l'utilisateur.

- Stratégies de sélection de la ressource : Bonita permet d'appliquer des filtres sur les ressources éligibles à exécuter une tâche.

  Par exemple, le filtre *Actor continuity* permet d'assigner une tâche à la ressource ayant exécuté la tâche précédente.

  Multiple Random et *Unique Random* permettent de sélectionner aléatoirement plusieurs candidats ou un candidat unique dans le groupe de ressources éligibles à exécuter la tâche.

  PerformerAssign permet d'utiliser une classe Java comme filtre.
- Moment de distribution : Que les noms des ressources soient connus dès la conception ou à l'exécution, l'assignation ne se fait que lorsque le processus est démarré. En effet, un utilisateur ne peut voir que les tâches appartenant à des instances de processus actives.

  L'assignation différée est partiellement possible car un utilisateur ayant les permissions nécessaires est capable de réassigner une tâche à un autre utilisateur, longtemps après l'activation de la tâche.

Ainsi, Bonita offre les tâches aux ressources. Plusieurs stratégies de sélection sont prédéfinies et il est possible de définir ses propres stratégies avec du code Java. Enfin, la distribution des tâches peut se faire lors de et après l'activation de l'instance de processus mais pas avant. Le produit obtient une note de 1 sur 2.

Enhydra Shark

- Offrir ou assigner le travail : Une tâche est offerte à un utilisateur ou à un groupe d'utilisateurs selon les correspondances entre participants et utilisateurs. Dans les deux cas, c'est au(x) utilisateur(s) d'accepter la tâche pour démarrer son exécution.
- Stratégies de sélection : Les ressources sont sélectionnées par rapport au tableau de correspondances rempli par l'administrateur des processus. Le système n'utilise donc pas de stratégie de sélection particulière.
- Moment de distribution : Les activités sont distribuées aux utilisateurs lors de leur

création. Éventuellement, une activité peut être réassignée ultérieurement à d'autres utilisateurs. La distribution avant l'instanciation du processus n'est pas prise en charge.

ES permet donc d'offrir une tâche à un utilisateur ou à un groupe d'utilisateurs. Le système se conforme au tableau de correspondance fourni par l'administrateur. Les tâches sont exclusivement distribuées à l'activation. Une note de 1 sur 2 est accordée pour ce critère.

6.4.3 Engagement des ressources à exécuter des tâches

jBPM

La console jBPM est une application web qui implémente l'authentification des utilisateurs. Un utilisateur authentifié obtient une liste des tâches qui lui sont offertes personnellement ou à un groupe dont il fait partie. L'utilisateur peut réclamer l'une des tâches afin de l'exécuter. L'exécution est automatiquement lancée après la réclamation. Si la tâche était offerte à un groupe d'utilisateurs, elle est alors retirée de la liste des tâches offertes aux autres membres du groupe.
L'utilisateur est capable de trier les tâches selon leurs noms, leurs états ou la date limite où elles doivent être achevées. Il a la liberté de démarrer n'importe quelle tâche, selon ses préférences.
Les méthodes de l'API correspondantes sont :

```
findGroupTasks(String identifiantUtilisateur)
findPersonalTasks(String identifiantUtilisateur)
takeTask(identifiantTache, identifiantUtilisateur)
```

Ainsi, l'API de jBPM permet aux ressources de s'engager à exécuter des tâches et d'organiser leur séquence de travail selon leurs préférences. Le produit obtient la note de 2 sur 2.

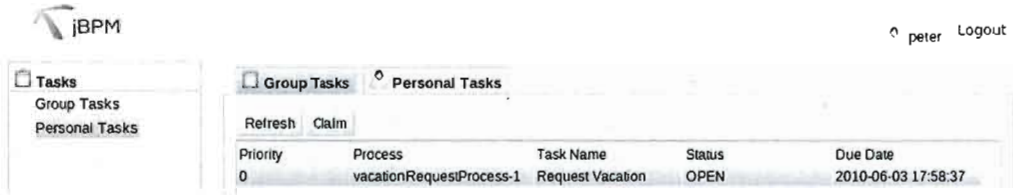


Figure 6.24 L'utilisateur authentifié est capable de réclamer une tâche

Bonita Open Solution

Les ressources gèrent les tâches qui leur sont allouées via l'interface User XP. Chaque tâche proposée à une ressource est déposée dans sa boîte de réception. La ressource a le choix entre l'exécuter, la consulter sans l'exécuter, la réclamer si elle est aussi proposée à d'autres ressources ou encore décliner l'offre.

Figure 6.25 Gestion des tâches par la ressource avec Bonita

Les tâches déposées dans la boîte de réception d'une ressource sont triées par leur date de création. Toutefois, étant donné que toutes les tâches sont consultables, une ressource est capable d'établir n'importe quel ordre d'exécution, selon ses préférences.
Bonita offre une grande autonomie aux ressources, leur laissant l'initiative d'exécuter les tâches selon leurs préférences. Le produit obtient une note de 2 sur 2.

Les tâches sont offertes aux utilisateurs qui les gèrent selon leurs préférences. Depuis l'interface d'administration, un utilisateur est capable de visualiser les éléments de sa liste de travail. Il peut alors décider de revendiquer une tâche (l'accepter) et démarrer son exécution. Il a le choix de poursuivre l'exécution d'une tâche non complétée (bouton *Complete*) ou encore de mettre à jour des variables de processus. L'utilisateur peut trier les éléments selon leur priorité, leur statut (accepté ou non), leur date de démarrage, la durée d'exécution déjà effectuée, etc. Le système lui laisse une autonomie totale sur les choix à faire, lui permettant notamment de réassigner des tâches qui lui sont offertes à d'autres utilisateurs.

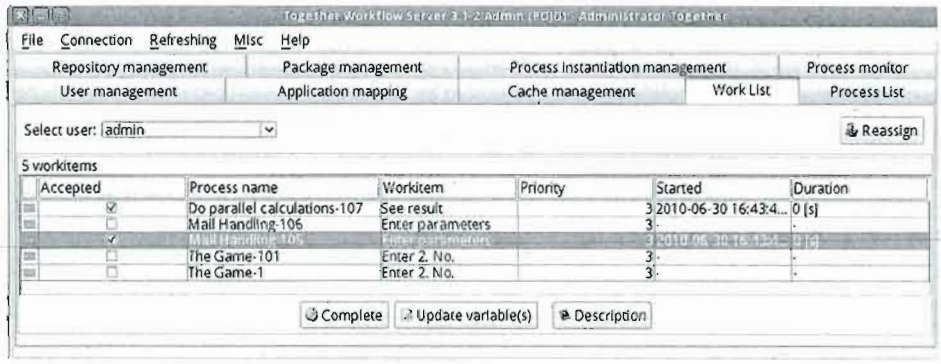


Figure 6.26 Gestion des tâches par la ressource avec ES

Ces fonctionnalités sont réalisées à partir de l'interface graphique en invoquant des méthodes de l'API d'ES tel que

```
addAcceptedByUser(WMSessionHandle sHandle, String nomUtilisateur, boolean
    lastRecordedOnly)
addRejectedByUser(WMSessionHandle sHandle, String nomUtilisateur, boolean
    lastRecordedOnly)
addStartedByUser(WMSessionHandle sHandle, String nomUtilisateur)
```

Enhydra Shark offre donc aux utilisateurs une grande autonomie et leur permet d'organiser leur travail selon leurs préférences. Le produit obtient une note de 2 sur 2.

6.4.4 Modification des attributions de tâches aux ressources

jBPM

- Délégation, Désallocation et Réallocation : jBPM n'offre pas aux ressources une commande prédéfinie pour déléguer une tâche. Il n'existe pas de commande propre à la désallocation et à la réallocation non plus. Par contre, ces fonctionnalités sont aisément réalisables avec du code Java, en utilisant les méthodes suivantes :

```
//Interface TaskService
assignTask(String taskId, String userId)
//Interface Assignable
addCandidateGroup(String groupId)
addCandidateUser(String userId)
setAssignee(String userId)
//Interface IdentityService
createMembership(String userId, String groupId, String role)
deleteMembership(String userId, String groupId, String role)
```

- Escalade : L'escalade est réalisée selon la procédure suivante
  - Démarrer un processus incluant une tâche
  - Créer la tâche avec une priorité de départ (par exemple NORMAL)
  - L'attribut *dueDate* de l'élément *task* indique la date butoir de la tâche. Au fur et à mesure que cette date butoir se rapproche, augmenter la propriété de la tâche.

  Les noms de priorités reconnus sont HIGHEST (2), HIGH (1), NORMAL(0), LOW (-1) et LOWEST (-2). Les autres niveaux de priorité peuvent être représentés avec leurs valeurs numériques, une valeur plus grande signifiant une priorité plus élevée.
  Par ailleurs, on pourrait définir une variable *escalate* qui est incrémentée à chaque déclenchement du chronomètre et réassigner la tâche à un autre utilisateur.
  La gestion de la variable de processus *escalate* pourrait être assurée par une classe implémentant l'interface *EventListener*.

```
<state name="nom de l etat">
   <on event="timeout">
     <timer duedate="10 minutes"/>
```

```
      <event-listener class="organisme.evenements.Escalade" />
    </on>
    <transition to="prochaine etape"/>
  </state>
```

La classe *Escalade* implémente la méthode *notify* de l'interface *EventListener* :

```
public class Escalade implements EventListener {
        public void notify(EventListenerExecution execution) {
        ...
        }
}
```

- Suspension/Reprise : La méthode *endProcessInstance* de l'interface *ExecutionService* permet de terminer une instance de processus. Lorsqu'une instance est terminée, son exécution s'arrête. Pour suspendre une tâche, on récupère une variable de type *Execution* pointant sur la tâche actuellement active. On met ensuite son état à *Execution.STATE_SUSPENDED.*

```
ProcessInstance pi = executionService.startProcessInstanceByKey("nom
    processus");
Execution executionDansTache = pi.findActiveExecutionIn("nom tache");
((OpenExecution)executionDansTache).setState(Execution.STATE_SUSPENDED);
```

Execution.STATE_SUSPENDED indique que l'exécution est temporairement suspendue. Les tâches suspendues allouées à un acteur humain n'apparaissent plus dans sa liste de tâches. Les chronomètres attachés à des tâches suspendues ne se déclenchent plus et l'exécution est verrouillée.

L'interface *OpenExecution* définissant la méthode *setState* est implémentée par les classes *ActivityExecution*, *EventListenerExecution* et *OpenProcessInstance* qui représentent respectivement l'exécution d'une activité, d'un *EventListener* et d'une instance de processus. Les états de ces trois types d'éléments peuvent donc être mis à *STATE_SUSPENDED.*

- Saut : La méthode *deleteTask(String taskId)* supprime une tâche sans l'avoir complétée et garde l'information journalisée pour la tâche dans la base de données. Si la tâche a été créée pendant l'exécution d'un processus, l'exécution reste active et le passage à la tâche suivante se fait en envoyant explicitement un signal à l'exécution en cours

avec la méthode *ExecutionService.signalExecutionById(String identifiantExecution)*. Une tâche ne peut être supprimée sans avoir été complétée que lorsqu'elle a été créée à l'exécution. Si on tente de supprimer une tâche définie dans le processus, le système lève une exception de type org.jbpm.api.JbpmException. Ceci constitue une limitation importante du système.

- Répétition : Définir une méthode qui prend en paramètre l'identifiant de l'instance de processus et qui ré-exécute un *job* référençant la tâche en cours. Un *job* est une unité de travail possédant une date butoir. Il s'agit généralement d'un chronomètre (*timer*) :

```
public void repeter(String idProcessus) {
        ProcessEngine moteurProcessus = new Configuration().setResource(
            "jbpm.cfg.xml").getProcessEngine();
        ManagementService serviceGestion = moteurProcessus.
            getManagementService();
        Job job = serviceGestion.createJobQuery().processInstanceId(id).
            uniqueResult();
        serviceGestion.executeJob(job.getId());
}
```

En résumé, jBPM ne propose aucune manière directe de définir la délégation, la désallocation, la réallocation, l'escalade, la suspension/reprise, le saut et la répétition de tâches. Hormis le saut de tâches, les autres fonctionnalités sont réalisables avec du code Java. Le produit obtient la note de 1 sur 2 pour ce critère, car aucune opération n'est directement prise en charge.

Bonita Open Solution

- Délégation, désallocation, réallocation : Elles sont réalisables soit avec l'API de Bonita soit via l'interface utilisateur User XP. Les méthodes suivantes permettent d'assigner une activité à un utilisateur ou de la proposer à un ensemble de candidats :

```
assign(ActivityInstanceUUID idInstanceActivite, Set<String> candidats)
assign(ActivityInstanceUUID idInstanceActivite, String idRessource)
assign(ActivityInstanceUUID idInstanceActivite)
```

```
unAssign(ActivityInstanceUUID idInstanceActivite)
unassignTask(ActivityInstanceUUID idInstanceActivite)
```

Figure 6.27 Réassigner une tâche à une autre ressource

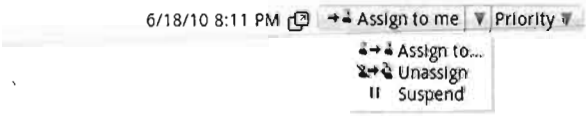


Figure 6.28 Se désengager d'une tâche

- Escalade : l'escalade est une forme de réallocation automatique. La réallocation est réalisable avec l'application User XP et avec l'API de Bonita. L'escalade est implémentée en définissant une date butoir pour l'exécution d'une tâche. Lorsque la date est dépassée, on programme la réallocation automatique de la tâche à une autre ressource et on augmente sa priorité. Ceci demande toutefois un effort de programmation.
- Suspension/Reprise : Réalisées avec l'API de Bonita ou avec l'interface web User XP. Les méthodes de l'API sont :

```
suspend(ActivityInstanceUUID idInstanceActivite, boolean allouerTache)
resume(ActivityInstanceUUID idInstanceActivite, boolean allouerActivite)
suspendTask(ActivityInstanceUUID idInstanceActivite, boolean
    allouerActivite)
```

```
resumeTask(ActivityInstanceUUID idInstanceAcitivte, boolean
    allouerActivite)
```

La suspension et la reprise sont réalisables intuitivement via l'interface User XP.

- Saut : On définit une branche parallèle aux étapes à sauter et ne contenant aucun élément. Les deux branches convergent vers une porte de type XOR (OU exclusif). Lorsque l'exécution d'une branche arrive à la porte XOR, les étapes de l'autre branche qui n'ont pas encore été exécutées sont ignorées. Les transitions vers chacune des branches peuvent être conditionnées par une expression qui détermine le chemin à suivre selon qu'on veuille exécuter les étapes ou les sauter. Le saut d'étapes n'est toutefois pas directement disponible à partir de l'interface User XP.
- Répétition : La répétition de l'exécution d'une étape n'est pas disponible à partir de l'interface User XP. Elle est toutefois réalisable en définissant une transition vers l'étape à répéter et en conditionnant la transition par une expression que l'on met a vrai si on veut effectuer une répétition.

Ainsi la délégation, la désallocation, la réallocation, la suspension et la reprise de tâches sont directement accessibles à partir de l'API de Bonita et de l'interface graphique User XP. Le saut d'étapes se fait en ajoutant des branchements parallèles. De même, l'escalade et la répétition ne sont pas directement exprimables et nécessitent un effort de programmation. Ce bilan mitigé vaut une note de 1 sur 2.

Enhydra Shark

- Délégation, désallocation, réallocation : Les fonctions sont directement accessibles à partir des applications d'administration Swing et web. Sous l'onglet liste de travail, un utilisateur est capable de gérer les tâches qui lui sont offertes ou dont il a commencé l'exécution. Pour déléguer une activité offerte et non acceptée, il appuie sur le bouton *Reassign* et sélectionne dans une liste d'utilisateurs celui auquel il désire déléguer l'activité.

  La réallocation se fait pour les activités acceptées et démarrées, de manière similaire à la délégation.

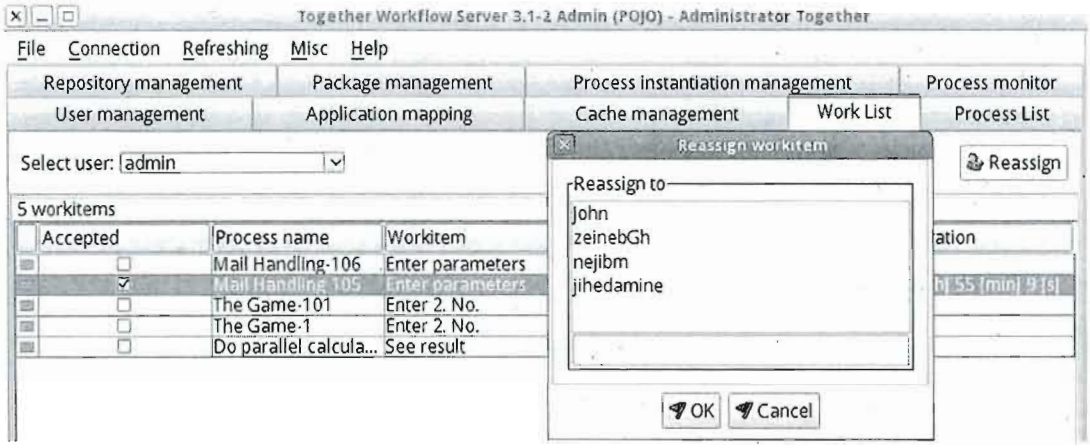


Figure 6.29 Délégation d'un item de travail avec ES

Pour se désallouer une activité, l'utilisateur décoche sa case *accepted*. L'activité peut alors être revendiquée par un autre utilisateur.

L'API d'ES implémente ces fonctionnalités. La méthode suivante effectue la réallocation :

```
void reassignWorkItem(WMSessionHandle shandle, String utilisateurSource,
    String utilisateurCible, String idInstanceProcessus, String
    idItemTravail)
```

- Escalade : L'escalade est réalisable en utilisant les dates butoirs, la réallocation et l'augmentation de priorité. La date butoir d'une activité est définie avec l'attribut *deadline*. Lorsque la date est atteinte, un agent doit être déclenché pour augmenter la priorité de l'activité et la réallouer à un autre utilisateur.

  L'agent pourrait utiliser les méthodes de l'API suivantes :

```
// Modifier la priorite
public void setPriority(int priority)
// Reallouer
reassignWorkItem(WMSessionHandle shandle, String utilisateurSource,
    String utilisateurCible, String indInstanceProcessus, String
    idItemTravail)
```

  Les niveaux de priorité sont des entiers naturels. La priorité la plus faible correspond à 0. Plus la valeur de l'entier est grande, plus la priorité de l'activité est élevée.

- Suspension/Reprise : Sous l'onglet *Process monitor* des applications d'administration, un bouton *Activity Management* ouvre une interface de gestion des activités. Elle contient notamment des boutons *Suspend* et *Resume* qui permettent respectivement de suspendre et de reprendre l'exécution d'une activité.

  La suspension/reprise d'une activité consiste à modifier son état. Les méthodes suivantes permettent la suspension d'une activité en cours d'exécution en mettant son état à *open.not_running.suspended* ou l'un de ses sous-états.

```
void suspend() throws CannotSuspend, NotRunning, AlreadySuspended
void suspend(WMSessionHandle shandle, String remoteInstanceUri, String
    processInstanceId, String workitemId)
```

  Les méthodes ci-dessous permettent de reprendre l'exécution d'une activité suspendue. L'état de l'activité passe de *open.not_running.suspended* à *open.running* ou l'un de ses sous-états.

```
void resume() throws CannotResume, NotSuspended
void resume(WMSessionHandle shandle, String remoteInstanceUri, String
    processInstanceId, String workitemId)
```

- Saut : Le saut d'activités n'est pas directement exprimable. Pour le réaliser, on peut créer une branche ne contenant aucun élément, parallèle aux étapes à sauter. Les deux branches convergent vers une porte de type *OU exclusif*. Les transitions vers chacune des branches doivent être conditionnées par une expression qui détermine le chemin à suivre selon qu'on veuille exécuter les activités ou les sauter en activant la branche parallèle vide.
- Répétition : Définir une transition de l'activité à répéter vers elle-même et conditionner l'activation de cette transition par une expression que l'on met a *vrai* à chaque fois que l'on veut répéter l'exécution.

ES permet directement la délégation, la réallocation, la désallocation, la suspension et la reprise d'items de travail via les applications d'administration et les méthodes de l'API. L'escalade, le saut et la répétition d'activités ne sont pas directement pris en charge mais peuvent être réalisés avec des solutions de contournement. Le produit obtient une

note de 1 sur 2.

6.5 Représentation et utilisation des données dans le flux

6.5.1 Définition des données pour un flux selon plusieurs contextes

jBPM

- Données liées à un processus : jBPM ne permet pas de définir des données globales. La portée des variables se limite aux instances de processus.
- Données locales à une instance de processus : Les variables de processus jBPM sont liées à une instance de processus.

  Le code suivant définit deux variables de processus et les affecte à une instance de processus :

```
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("variableProcessus", "a1");
ProcessInstance instanceProcessus = executionService.
    startProcessInstanceByKey("processus", variables);
```

- Données locales à une tâche : Les variables d'une tâche sont créées ou redéfinies avec la méthode :

```
setVariables(String identifiantTache, Map<String,Object> variables)
```

  L'exemple suivant récupère une tâche attribuée à la ressource *res1* dans l'instance de processus en cours et lui affecte des variables :

```
Task a = taskService.findPersonalTasks("res1").get(0);
variables = new HashMap<String, Object>();
variables.put("variableProcessus", "b1");
variables put("variableLocaleTache", "b3");
taskService.setVariables(a.getId(), variables);
```

  En réalité, jBPM crée des variables de processus. Pour le vérifier, on affiche les valeurs des variables de tâche et des variables de processus après avoir exécuté les instructions précédentes :

```
Tache A: {variableLocaleTache=b3, variableProcessus=b1}
Processus: {variableLocaleTache=b3, variableProcessus=b1}
```

Pour définir des variables uniquement visibles aux tâches, il faudrait utiliser des tâches de type script ou Java et y déclarer des variables locales.

- Données locales à un bloc de tâches : Si le bloc de tâches peut former un sous-processus, il est possible de définir des variables locales à ce sous-processus.

  Un sous-processus est référencé de la manière suivante :

```
<process name="processus-parent">
  <start>
    <transition to="sous-processus" />
  </start>

  <sub-process name="sous-processus">
...
</process>
```

  Le sous-processus est défini comme un processus régulier :

```
<process name="sous-processus">
  <start>
   ...
</process>
```

  Des variables locales sont attachées au sous-processus comme suit :

```
ProcessInstance processInstance = executionService.
    startProcessInstanceByKey("sous-processus", variables);
```

  Actuellement, une construction nommée *group* a été introduite à titre expérimental dans jBPM. Un groupe est similaire à un sous-processus, sauf qu'il est défini à l'intérieur du processus parent. Il n'est présentement pas possible d'attacher des variables à un groupe.

- Données d'environnement : L'accès aux données d'environnement se fait avec du code Java. En effet, le langage fournit plusieurs bibliothèques d'accès aux sources de données externes tel que les bases de données relationnelles, les services web, etc.

jBPM permet de définir des variables locales à une instance de processus. Il est partiellement possible de créer des variables locales à une tâche et à un bloc de tâches. Les variables d'environnement sont accessibles via du code Java. Toutefois, jBPM ne permet pas la création de variables globales à toutes les instances d'un processus. Le produit obtient une note de 1 sur 2.

Bonita Open Solution

- Données locales aux tâches : Les étapes d'un processus peuvent posséder des données locales. Dans l'éditeur Bonita, l'onglet *Data* d'une étape permet de manipuler ses données. Les données locales sont promues en données d'instance de processus en cliquant sur le bouton *Promote* de l'éditeur ou en utilisant la méthode correspondante de l'API. Les données sont de type numérique, chaîne de caractère, booléen, liste ou n'importe quel type Java ou personnel importé dans Bonita sous forme d'archive jar.
- Données d'une instance de processus : La gestion des données locales à une instance de processus est similaire à celle des données locales aux tâches. Des méthodes correspondantes existent dans l'API de Bonita.
- Données locales à un bloc : Un sous-ensemble de tâches interconnectées est rassemblé dans un même sous-processus. Les données sont définies localement au sous-processus.
- Données de la définition de processus : La portée des données de processus se limite aux instances : Si on déclare une variable de processus initialisée à une même valeur pour toutes les instances d'un processus, lorsque cette valeur est modifiée dans une instance, le changement n'est pas répercuté dans les autres.
- Données de l'environnement : Les connecteurs permettent de récupérer toute sorte de données de l'environnement (Service web, répertoire LDAP, document partagé sur Alfresco...).

Bonita permet de définir des données locales aux tâches, à des blocs de tâches (si elles sont interconnectées) et aux instances de processus. Toutefois, il n'est pas directement possible de définir des données globales à toutes les instances de processus. Une note de 1 sur 2 est accordée sur ce critère.

Enhydra Shark

- Données locales aux tâches : Avec Enhydra Shark, les données sont déclarées au niveau des processus et non des activités. Toutefois, l'attribut étendu OVERRIDE_PROCESS_CONTEXT utilisable au niveau des activités permet de redéfinir une variable provenant de l'instance de processus avec une nouvelle valeur passée en paramètre. La nouvelle valeur est effective dans le contexte de l'activité. Il est possible de redéfinir les valeurs de plusieurs variables en les séparant par des virgules (var1 :val1,var2 :val2...). Par ailleurs, les tâches de type procédures JavaScript et les agents peuvent posséder leur variables locales.
- Données d'une instance de processus : Les valeurs des variables de processus et de paquetages de processus sont locales à chaque instance de processus.

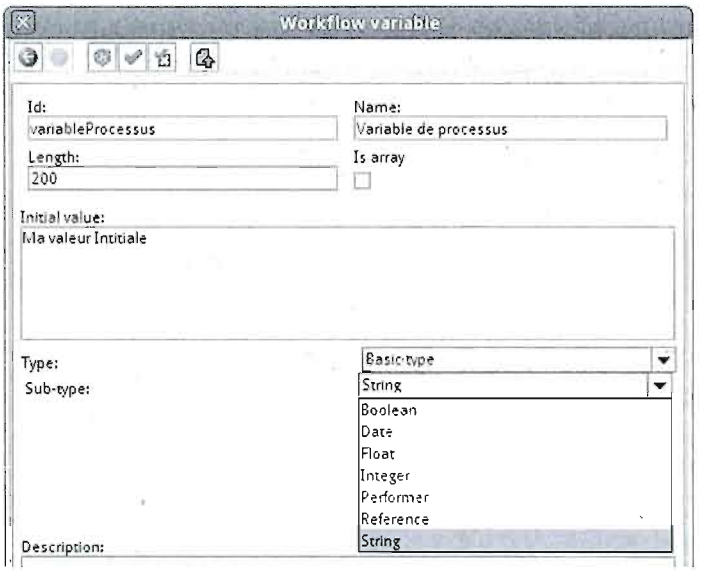


Figure 6.30 Définition d'une variable de processus ES

- Données locales à un bloc : Définissables avec les variables de sous-flux (*SubFlow*). Ces données ne sont pas visibles au processus appelant et n'existent que dans le contexte du sous-flux. Par ailleurs, les blocs d'activités peuvent déclarer des variables, mais elles sont globalement visibles dans l'instance de processus.

- Données de la définition de processus : Même s'il est possible de déclarer des variables de paquetage, leurs valeurs varient d'une instance de processus du paquetage à l'autre. Il est impossible de les utiliser comme données partagées par plusieurs instances de processus.
- Données de l'environnement : Les agents fournissent une interface vers des sources de données externes. Un agent Java ou JavaScript, par exemple, est capable d'accéder à des données du système d'exploitation, stockées dans une base de données ou encore dans des fichiers XML.

ES permet de définir des données locales aux instances de processus et aux blocs de tâches. Une tâche peut avoir ses valeurs locales pour des données de l'instance de processus. De plus, il est possible d'accéder à des données de l'environnement. Le produit n'obtient que la note de 1 sur 2 car il ne propose pas de solution directe pour partager les données entre plusieurs instances d'un même processus.

6.5.2 Transfert de données entre composants d'un processus

jBPM

- Tâche à tâche : Les variables de processus sont partagées entre plusieurs tâches d'un même processus. Les tâches ne sont pas directement capables de s'envoyer des données.
- Transfert de données vers une tâche à instances multiples : jBPM ne prend pas en charge la notion d'instances multiples d'une tâche.
- Instance de processus à instance de processus : Réalisé indirectement en passant par exemple par une base de données externe. Une tâche personnalisée se chargerait alors d'écrire ou de lire les données à transférer dans la base de données.

Ainsi, jBPM permet de partager des données entre tâches. Le système ne gère pas le concept d'instances multiples. Le transfert inter-processus est indirectement possible. On attribue la note de 1 sur 2 pour ce critère.

- Instance de processus à instance de processus : Pour envoyer des données d'un processus P1 à un processus P2, on définit un événement de type *Message Throw* dans le processus P1 et un événement de type *Message Catch* dans le processus P2. Dans l'onglet *Messages* du message *Throw*, on crée un message destiné au message *Catch* du processus P2. On définit ensuite un ensemble de données à transférer dans le message. Les valeurs des données sont initialisées avec les valeurs des variables du processus P1. On déclare ensuite des variables de processus dans P2 qui vont récupérer les données transférées depuis le processus P1. Enfin, on ajoute un connecteur de type *Set Variable* au message *Catch* de P2. Le connecteur affectera les valeurs lues depuis le message envoyé par P1 aux variables du processus P2.

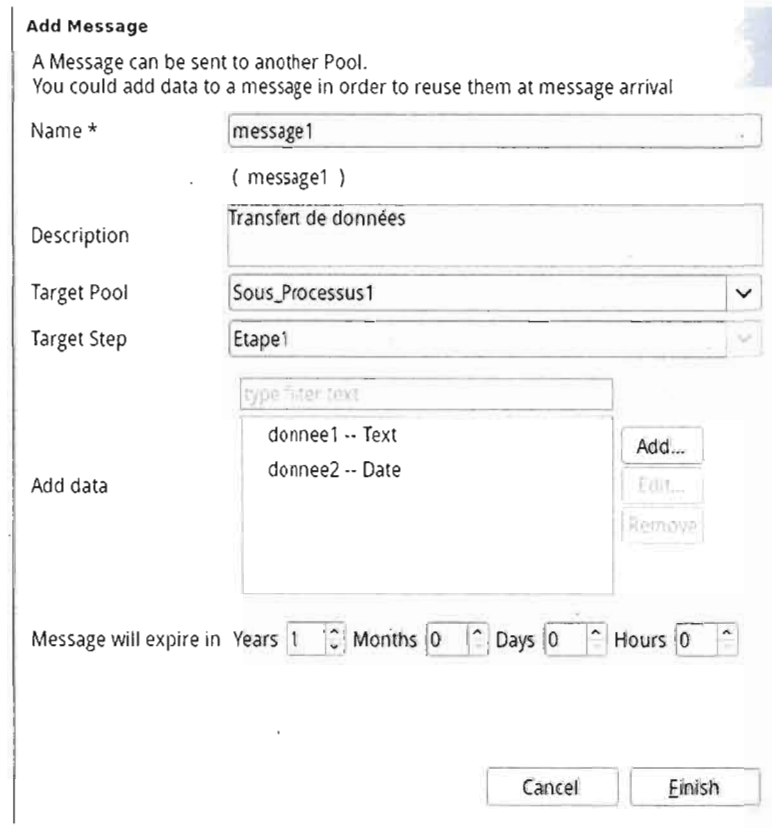


Figure 6.31 Envoi de données dans un message

Le partage de données entre instances de processus peut se faire via des sources de données externes, accessibles aux deux processus, notamment via des connecteurs.

- Tâche à tâche : Les tâches présentes dans des couloirs différents ne sont capables de s'envoyer directement des données via des événements de type *Message Throw* et *Message Catch*. Les données qui leurs sont locales doivent être promues en données de processus afin de réaliser la procédure expliquée pour les transferts de données entre processus.

  L'alternative serait de partager des données avec des variables de processus.

- Transfert de données vers une tâche à instances multiples : Bonita permet de définir des données locales aux instances d'une tâche multi-instanciée. Le connecteur *MultiInstantiator* est une classe Java qui définit la méthode

```
execute(QueryAPIAccessor accessor, ProcessInstanceUUID UUIDinstance,
    String idActivite, String idIteration)
```

  Cette méthode prend en paramètre l'identifiant d'une instance (*idIteration*). Elle est donc capable d'affecter des valeurs spécifiques à chacune des instances multiples. Le nombre d'instances peut être connu à l'exécution.

Bonita offre des mécanismes de transfert de données par messages. L'échange entre tâches est réalisable avec des variables partagées. Le transfert ciblé vers chacune des instances multiples d'une tâche est pleinement satisfait. Le produit obtient la note de 2 sur 2.

Enhydra Shark

- Tâche à tâche : Les activités au sein d'un même processus partagent des données en passant par les variables de processus.
- Instance de processus à instance de processus : ES ne permet pas un transfert direct des données entre instances de processus. La fonctionnalité est réalisable par l'intermédiaire d'une source de données externes. Les données ne sont toutefois pas destinées à une instance de processus en particulier, mais sont accessibles à tous les processus qui communiquent avec la source de données.

- Transfert de données vers une tâche à instances multiples : L'implémentation ES du patron d'instances multiples permet de transférer des données à toutes les instances multiples ou de cibler une instance en particulier.

```
<xpdl:DataFields>
  <xpdl:DataField Id="donneescommunes">
  ...
  <xpdl:DataField Id="donneesInstance1">
  ...
  <xpdl:DataField Id="donneesInstance2" IsArray="FALSE">
  ...
<xpdl:Activity Id="S1" Name="S1">
 <xpdl:SubFlow Execution="ASYNCHR" Id="Instance1">
    <xpdl:ActualParameters>
       <xpdl:ActualParameter>donneescommunes</xpdl:ActualParameter>
       <xpdl:ActualParameter>donneesInstance1</xpdl:ActualParameter>
...
\end{itemize}
```

  La contrainte de cette solution est que le nombre d'instances doit être connu à la conception. Pour une tâche multi-instanciée dont le nombre d'instances est inconnu à la conception, il n'est pas directement possible de cibler les données vers une instance particulière.

ES permet de partager des données entre tâches et entre instances de processus. Le transfert à une instance d'une tâche multi-instanciée n'est possible que lorsque le nombre d'instances est connu à la conception. Le produit obtient la note de 1 sur 2.

6.5.3 Transfert de données entre processus et environnement

jBPM

- Instance de processus - Environnement : Une instance de processus peut échanger des données avec l'environnement externe en passant par des méthodes Java qui manipulent les variables de processus. Plusieurs bibliothèques Java permettent l'interaction

avec des sources de données externes.

- Tâche - Environnement : Les variables de tâches jBPM sont en réalité des variables de processus. Pour qu'une tâche échange des données avec l'environnement, elle devra passer par une méthode Java qui déclare des variables locales dont les valeurs sont passées ou récupérées de l'environnement externe.
- Processus - Environnement : La portée des variables jBPM se limite à une instance de processus. Toutefois, il est possible par programmation de s'assurer que la même valeur est partagée par toutes les instances du processus. Par conséquent, l'échange entre processus et environnement externe est indirectement réalisable.

jBPM permet indirectement le transfert de données entre l'environnement externe et les tâches. Le transfert avec une définition de processus n'est pas pris en charge. Le produit obtient une note de 1 sur 2.

Bonita Open Solution

- Instance de processus - Environnement : Un processus peut envoyer et recevoir des données de l'environnement externe en utilisant des événements de type *Message Throw* et *Message Catch*, respectivement. Les messages sont envoyés à un couloir ou à un sous-processus qui effectue le lien avec le système externe via des connecteurs. Les données reçues de l'environnement externe par le processus sont affectées aux variables du processus.

  L'assistant de définition des connecteurs permet d'établir une correspondance entre les résultats récupérés de l'environnement et les variables de processus. Par exemple, les résultats des requêtes SQL de sélection peuvent être affectés à des variables du processus. L'opération est possible pour tous les types de connecteurs qui produisent un résultat.
- Tâche - Environnement : La procédure utilisée pour transférer des données entre instance de processus et environnement n'est pas possible pour des données locales aux tâches. Celles-ci doivent être promues en variables de processus pour être visibles aux messages *Throw* et *Catch*. L'alternative serait de définir un connecteur dans

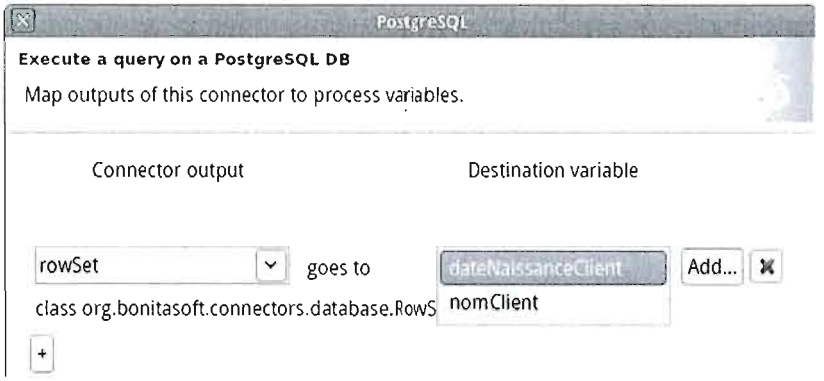


Figure 6.32 Correspondance entre résultats du connecteur et variables du processus

la tâche et de réaliser les correspondances entre données de la tâche et données de l'environnement.

- Processus - Environnement : Étant donné que Bonita ne permet pas directement de définir des données globales à toutes les instances d'un processus, ce mode de transfert n'est pas directement réalisable.

Bonita obtient une note de 1 sur 2 car il ne satisfait pas pleinement les patrons de transfert avec l'environnement.

Enhydra Shark

Les agents permettent d'échanger des données avec une large palette d'applications externes. Les données peuvent être transférées via des paramètres formels qui sont passés à l'agent lors de son invocation. Les références externes constituent un moyen alternatif aux paramètres formels. Elles utilisent des définitions externes d'entités.

- Instance de processus - Environnement : l'agent envoie ou retourne des données à l'instance de processus qui le définit.
- Tâche - Environnement : Les données dont les valeurs sont envoyées ou récupérées de l'environnement externe sont redéfinies localement à une tâche.
- Processus - Environnement : ES ne permet pas de définir de variables globales à toutes les instances d'un processus. Ainsi, la valeur lue ou passée à l'environnement externe

peut être différente d'une instance de processus à l'autre.

Enhydra Shark obtient la note de 1 sur 2 pour ce critère.

6.5.4 Prise en charge de divers mécanismes de transfert

jBPM

- Passage par copie : Les variables de processus définies dans jBPM sont transférées par copie. Chaque élément du flux reçoit une copie des valeurs des variables. Les modifications effectuées par un élément du flux sont copiées dans la variable de processus globale lorsque l'élément a terminé son exécution. Ce mécanisme pose le problème des mises à jour perdues lorsque des éléments s'exécutent en parallèle car celui qui terminera son exécution le dernier va écraser les anciennes valeurs de la variable.
- Passage par valeur : en utilisant du code Java, on est capable de passer des données par valeur. En effet, les paramètres des méthodes sont passés par valeur dans le langage Java. Les valeurs représentent les références des objets.
- Passage par référence : jBPM n'utilise pas le mécanisme de passage de données par référence.
- Transformation des données : Les données peuvent être transformées par des méthodes (Java ou langage de script) à l'initialisation ou à la fin de l'exécution d'une tâche. Le déclenchement de la méthode se fait via un *EventListener* lié à l'événement *node-enter* si la transformation est effectuée à l'initialisation d'une tâche ou à l'évènement *node-leave* si la transformation se fait à la complétion de la tâche.

```
<task name="tache1">
  <on event="node-enter">
    <event-listener class="org.ClasseDeTransformation"
                    method="methodeDeTransformation" />
  </on>
</task>
```

Il est possible de transformer les données dans des sous-processus en utilisant les paramètres d'entrée et de sortie :

```
<sub-process name="review">
    <parameter-in var="donneeInitiale" subvar="
        copieDonneInitialeDansSousProcessus" />
    <parameter-out var="donneeTransformee" subvar="
        copieDonneeTransformeeDansSousProcessus" />

    <transition to="prochaine etape" />
</sub-process>
```

Alternativement, on peut définir une activité de type *rules-decision* qui déclencherait des règles Drools de transformation de données.

Ainsi, jBPM permet le transfert de données par copie et par valeur (via le langage Java). La transformation des données peut être réalisée avec plusieurs mécanismes. Le fait que jBPM ne permette pas de passer des données par référence réduit sa note à 1 sur 2.

Bonita Open Solution

- Passage par valeur : Les messages utilisent un mécanisme de passage par valeur. La valeur d'une variable de processus est passée dans une variable du message de type *Throw* au message récepteur de type *Catch*.
- Passage par référence : Les variables de processus sont accédées avec leurs références. Ceci permet de partager des données globales et de se passer de la procédure de transfert de données.
- Passage par copie : La correspondance entre données d'un processus et données récupérées via un connecteur, ainsi que la correspondance entre données d'un processus et d'un sous-processus se font en copiant les attributs d'une entitée à l'autre.
- Transformation des données : Il est possible de transformer des données à l'entrée et à la sortie d'une étape du processus en définissant des connecteurs de type script Groovy ou règle Drools, par exemple, qui se déclenchent respectivement au démarrage et à la fin de l'exécution d'une étape.

Bonita permet d'utiliser les trois mécanismes de transferts de données et d'appliquer des transformations sur les données. Il obtient la note de 2 sur 2.

Enhydra Shark

- Passage par valeur : Les données sont passées de et vers les sous-flux (*SubFlow*) ou sous-processus avec des paramètres.

```
<xpdl:Activity Id="S1" Name="S1">
 <xpdl:SubFlow Execution="ASYNCHR" Id="Instance1">
    <xpdl:ActualParameters>
       <xpdl:ActualParameter>donneescommunes</xpdl:ActualParameter>
```

  Les paramètres ne sont pas utilisés pour transférer des données entre d'autres types d'activités car les données sont globales à l'instance de processus.
- Passage par référence : Enhydra Shark n'utilise pas le mécanisme de passage par référence.
- Passage par copie : Le passage par copie est utilisé lors de l'invocation de sous-processus. Les valeurs des variables sont copiées vers les variables de sous-processus par l'intermédiaire de paramètres ou de copies locales des variables globales. Pour les invocations synchrones, le sous-processus copie les valeurs de ses variables et les passe comme paramètres vers le processus appelant. Pour les invocations asynchrones, les valeurs des variables ne sont pas copiées vers le processus appelant afin d'éviter des problèmes de mise à jour concurrente.
- Transformation des données : Des tâches automatiques de types agents peuvent être définies après l'activité qui produit les données transformées. Les agents reçoivent les données en paramètre et retournent les valeurs transformées. Par exemple, l'agent XSL applique des transformations XSL à une chaîne de caractères, un tableau d'octets ou une variable XML passée en paramètre et retourne le résultat après transformation. Il est possible de définir des procédures de transformation JavaScript directement dans les définitions de processus, sans passer par les agents.

Ainsi, ES utilise les mécanismes de passage par valeur et par copie. Les agents et les procédures JavaScript peuvent réaliser des transformations sur les données. Une note de 1 sur 2 est accordée car ES n'utilise pas le mécanisme de transfert par référence.

6.5.5 Utilisation des données comme conditions

jBPM

jBPM permet de définir des préconditions et des postconditions avec les deux langages actuellement pris en charge (jPDL et une implémentation de BPMN).

- Avec jPDL, on peut restreindre les transitions entre éléments de flux par des conditions. L'expression de la condition peut être écrite en Java ou avec un langage de script dont il existe un moteur conforme à la spécification JSR-223 (Python, Ruby, Groovy et BeanShell, awk...) :

```
<decision name="evaluer demande">
    <transition to="soumettre demande">
      <condition lang="java" expr="#{contenu=="acceptable"}" />
    </transition>
    <transition to="reevaluer">
      <condition expr="#{content=="moyen"}" />
    </transition>
    <transition to="rejeter" />
</decision>
```

- Avec l'implémentation de BPMN, le principe est similaire. Seule la syntaxe change :

```
<sequenceFlow id="identifiantTransition">
  <conditionExpression xsi:type="tFormalExpression">
     ${montant >= 500}
  </conditionExpression>
</sequenceFlow>
```

Ainsi, jBPM permet directement d'exprimer des conditions et obtient la note de 2 sur 2 pour ce critère.

Bonita Open Solution

L'exécution d'une étape peut être contrainte en conditionnant les transitions à l'entrée et à la sortie de l'étape. L'éditeur Groovy de Bonita facilite la création d'expressions Groovy utilisées comme conditions.

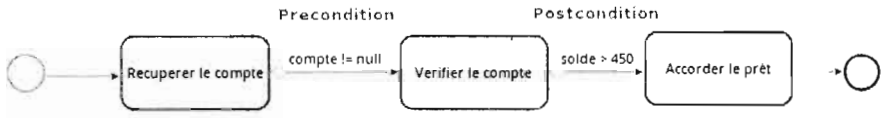


Figure 6.33 Précondition et postcondition pour l'étape Vérifier le compte

Le produit obtient une note de 2 sur 2 pour ce critère.

Enhydra Shark

L'activation de transitions peut être contrainte par des conditions. L'exemple suivant conditionne l'activation de la transition de l'activité *A* vers l'activité *B* par une valeur égale à *val* de la variable *var1*.

```
<xpdl:Transitions>
   <xpdl:Transition From="A" Id="Tra1" To="B">
      <xpdl:Condition Type="CONDITION">var1=="val"</xpdl:Condition>
...
```

Les conditions peuvent être complexes. L'expression suivante est vraie si la variable *var1* contient la lettre *D*.

```
  <xpdl:Transition From="A" Id="Tra1" To="B">
     <xpdl:Condition Type="CONDITION">
        var1.indexOf('D') != -1
...
```

L'appellation précondition ou postcondition est relative à une activité. Si l'activité est la destination de la transition conditionnée, alors il s'agit d'une précondition. Si elle en

est la source, alors il s'agit d'une postcondition. Pour l'exemple précédent, *var1 contient la lettre D* est une précondition pour l'activité B et une postcondition pour l'activité A. Enhydra est donc capable de définir des préconditions et des postconditions. Il obtient la note de 2 sur 2.

6.6 Accessibilité aux experts métier

6.6.1 Définition graphique des processus

jBPM

jBPM fournit un plugiciel *eclipse* appelé GPD (Graphic Process Definition) qui permet d'éditer les fichiers de définition de flux dans un environnement graphique. Les éléments de flux sont accessibles à partir d'une palette de composants. La vue *eclipse* des propriétés affiche les attributs relatifs à l'élément sélectionné dans le diagramme (voir figure 6.48).

Un onglet *source* permet de basculer vers le code de définition de flux, automatiquement généré à partir de la modélisation graphique. L'attribut optionnel "g" définit la position d'un élément dans la représentation graphique.

En plus de l'éditeur graphique GPD, jBPM fournit une application web, créée en collaboration avec la société Signavio [14]. L'outil permet de définir les attributs des éléments du flux et d'évaluer la validité des processus en signalant les erreurs éventuelles et leurs causes. L'application Signavio a l'avantage d'être indépendante de l'environnement *eclipse* et accessible à partir d'un navigateur web.

Les experts métier pourraient modéliser les processus via l'application Signavio et les partager sur le réseau local de l'organisme.

On note que, pour le moment, les éditeurs GPD et Signavio ne peuvent créer des modèles BPMN. Cette fonctionnalité est toutefois prévue pour les futures versions du produit.

La définition graphique des processus est donc prise en charge par jBPM, qui obtient

14. http ://www.signavio.com/en.html (dernière consultation le 09-11-10)

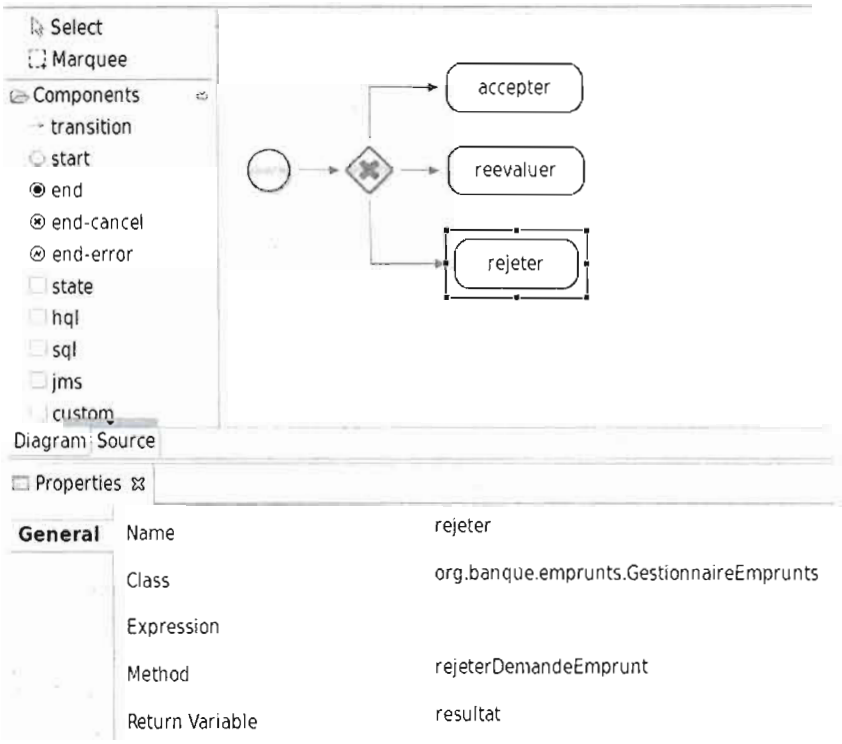


Figure 6.34 Définition graphique de flux avec le plugiciel *eclipse*

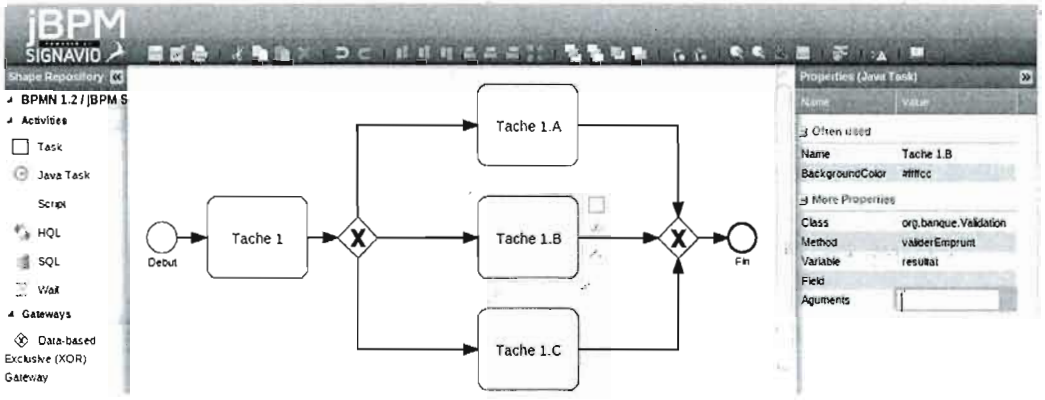


Figure 6.35 Application web Signavio de modélisation de flux

une note de 2 sur 2.

Bonita Open Solution

La définition des processus dans Bonita repose exclusivement sur des diagrammes conformes au standard BPMN 2.0. Lorsqu'on importe des processus définis avec jBPM ou du code XPDL ou BPMN (fichiers XMLs d'extension .xpdl ou .bpmn), Bonita génère automatiquement les diagrammes correspondants. À partir des représentations graphiques, on est capable de sélectionner des étapes, d'ajouter des données, d'attacher des connecteurs, etc. De plus, lorsqu'une étape est sélectionnée, l'ensemble des actions possibles est représenté graphiquement (lier à une autre étape, à un événement, à une porte de convergence...). Si le type d'un élément est modifiable (par exemple changer une étape automatique en une étape humaine), les différentes possibilités sont affichées. On peut ainsi modifier la nature de l'élément directement dans le diagramme, en cliquant sur le nouveau type. De même, lorsqu'on désire créer un élément d'une certaine nature (par exemple un événement), l'ensemble des événements possibles (Message *Catch*, *Throw*, Fin de processus...) est affiché et on sélectionne graphiquement le type désiré.

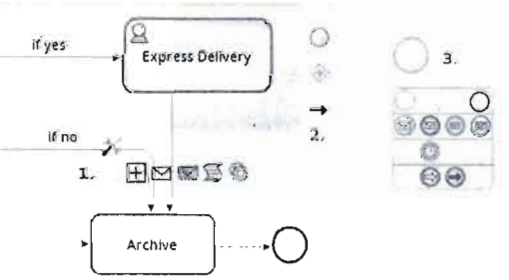


Figure 6.36 L'éditeur graphique liste les types d'éléments possibles

Dans la figure 6.36, on modifie le type de l'étape en une étape humaine (1.), on choisit ensuite de créer une transition vers un événement à partir de cette étape (2.). Enfin, on sélectionne un type d'événement parmi les neuf types possibles (3.).

L'éditeur Bonita affiche dans la fenêtre *Details* l'ensemble des attributs pertinents pour l'entité sélectionnée, groupés par catégories. L'onglet *General* permet de définir le nom de l'entité et de sélectionner son type dans une liste déroulante. L'onglet *Advanced* per-

met entre autres de cocher un bouton radio *Is-MultiInstanciated* ou *Is-Looped*. L'onglet *Data* offre des fonctions de gestion des données locales à l'entité sélectionnée (ajout, modification, promotion d'une variable locale...). Les onglets *Actors* et *Connectors* permettent respectivement de gérer les participants et les connecteurs de l'entité. D'autres onglets peuvent exister, selon le type de l'entité sélectionnée.
L'éditeur graphique Bonita est un outil intuitif et puissant, gérant le processus complet de définition de processus. Une note de 2 sur 2 est accordée sur ce critère.

Enhydra Shark

Together Workflow Editor est l'éditeur graphique d'Enhydra Shark. Il permet de créer et modifier des fichiers XPDL de définitions de processus. L'éditeur fournit une palette de composants contenant les différents types de participants, d'activités et de transitions. Les éléments sont instanciés simplement en sélectionnant le type correspondant dans la palette et en les plaçant dans la représentation graphique du processus.

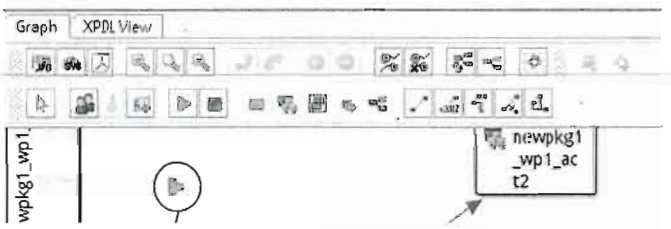


Figure 6.37 Palette de composants de l'éditeur TWE

Les diagrammes peuvent être agrandis, réduits et exportés sous format PDF, JPG ou SVG. La configuration des représentations graphiques des processus se fait à travers un fichier *togwegraphcontroller.properties*.
Il est possible de sélectionner un ensemble d'éléments du graphe en appuyant sur la touche SHIFT ou CTRL tout en sélectionnant individuellement les éléments. Un clic droit sur la sélection affiche un menu contextuel. Par exemple, pour une transition, le menu permettra d'en modifier l'apparence, de la supprimer ou encore d'afficher l'ensemble de ses propriétés dans une boîte de dialogue à onglets multiples. Chaque onglet

permet de gérer une catégorie particulière des propriétés de l'élément sélectionné.
Une vue *Properties* en bas de la représentation graphique du processus expose les propriétés de l'élément actuellement sélectionné. Les propriétés principales d'un élément sont également listées lorsque le curseur se déplace dessus.

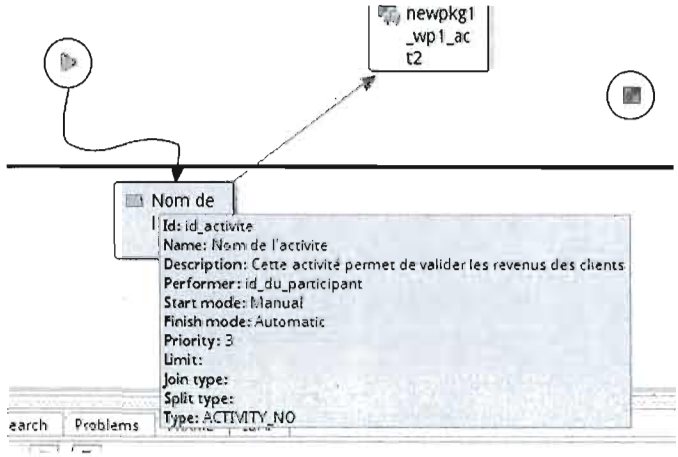


Figure 6.38 Fenêtre popup des propriétés d'un élément

Une vue *XPDL View* montre le code XPDL pour l'élément sélectionné. Ceci permet de visualiser rapidement le bout de code relatif à une partie du processus.
Par ailleurs, la vue *Navigator* présente la hiérarchie du modèle XPDL. Au plus haut niveau se trouvent les paquetages. En étendant le nœud d'un paquetage, on obtient la liste des processus qu'il contient. En descendant encore dans la hiérarchie, on parcourt les éléments définis pour un processus donné.
L'éditeur peut importer des fichier XPDL, possiblement définis avec d'autres produits conformes à XPDL. Le diagramme correspondant est automatiquement généré.
Ainsi, l'éditeur TWE permet de définir et de modifier les processus graphiquement. L'édition est facilitée par des assistants. Les représentations graphiques sont automatiquement générées pour les définitions de processus conformes au format XPDL. Le produit obtient une note de 2 sur 2.

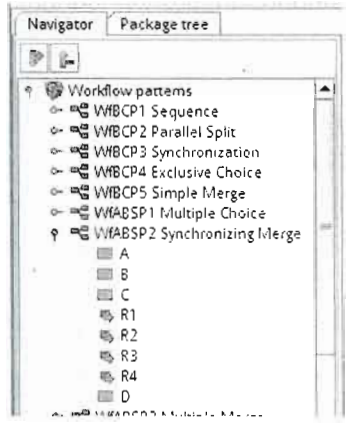


Figure 6.39 Vue *Navigator* de l'éditeur TWE

6.6.2 Définition assistée des activités

jBPM

Les fichiers de définition de flux jPDL et BPMN sont des fichiers XML obéissant à une syntaxe. La spécification de la syntaxe permet aux éditeurs tel qu'*eclipse* d'assister l'utilisateur avec l'auto-complétion. Cette fonctionnalité est également disponible pour la programmation de tâches en Java et pour certains langages de script (Groovy, Ruby...).

L'application jBPM assiste l'utilisateur dans la création des formulaires échangés avec les acteurs humains. En effet, la console jBPM réutilise la bibliothèque *freemarker*[15] pour définir les fichier .ftl. Les noms des champs de texte du formulaire .ftl sont transposés en variables de processus et vice-versa. Un champ *outcome* permet de continuer l'exécution après avoir validé le formulaire.

```
<html>
  <body>
    <form action="${form.action}" method="POST" enctype="multipart/form-
        data">
```

15. http ://freemarker.sourceforge.net/ (dernière consultation le 09-11-10)

```
      <h3>Votre client , ${employee_name} , aimerait contracter un pret.</
          h3>
      Nombre de jours: ${number_of_days}<br/>
      <hr>
      En cas de refus , veuillez fournir une raison:<br/>
      <input type="textarea" name="raison"/><br/>
      <#list outcome.values as transition>
          <input type="submit" name="outcome" value="${transition}">
      </#list>
    </form>
  </body>
</html>
```

À partir de ces quelques lignes de code, on obtient le formulaire suivant.

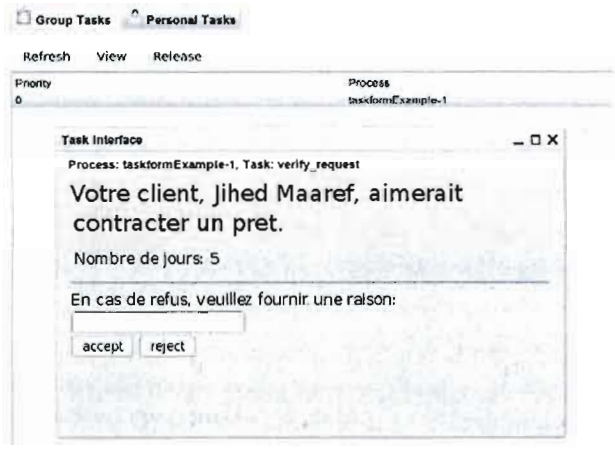


Figure 6.40 Formulaire généré pour la console jBPM

La valeur du champ *raison* est stockée comme variable de processus.

D'autre part, l'application web Signavio dispose d'un vérificateur de syntaxe.

jBPM ne fournit pas de points d'intégration avec des logiciels externes tel que les suites bureautiques et les logiciels de gestion de documents. Il s'agit d'un handicap important, vu que les flux de travail en entreprise font appel à des applications externes. Pour cette raison, le produit n'obtient que 1 sur 2.

En plus des assistants de définition des processus, Bonita fournit d'autres assistants pour la quasi-totalité des fonctionnalités du système.

- Assistants de définition des ressources : La gestion des ressources bénéficie de ses propres assistants. Si, par exemple, on récupère les noms des ressources d'un dépôt LDAP, l'assistant affiche un formulaire où saisir le nom de l'hôte, le numéro de port, le protocole utilisé, etc. En fonction du type de ressource, des formulaires adéquats sont affichés. Il en est de même pour les filtres sur les groupes, dont plusieurs sont prédéfinis (ressource ayant exécuté la tâche précédente, sélection aléatoire dans un groupe...).

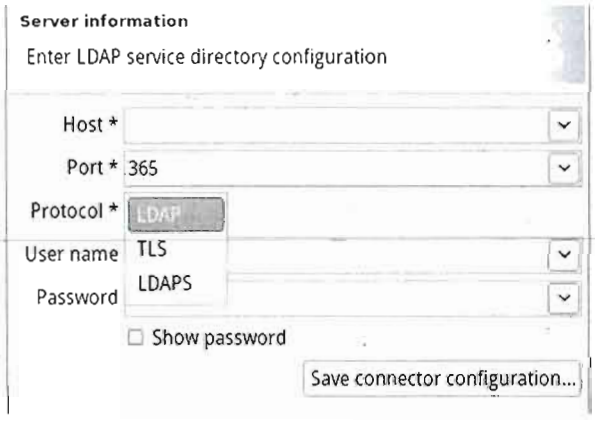


Figure 6.41 Assistant de configuration d'un dépôt LDAP

- Assistants d'édition des expressions Groovy et du code Java : Bonita fournit des éditeurs Groovy et Java qui réutilisent les fonctionnalités de l'IDE *eclipse* (coloration syntaxique, auto-complétion, etc). Les variables de processus et les méthodes des bibliothèques Java et Groovy incluses dans le projet Bonita sont listées dans des listes déroulantes et aisément accessibles.
- Assistants de définition des connecteurs : Les assistants guident l'utilisateur dans la configuration des connecteurs, en fonction de leurs types.
- Assistant de définition des interfaces de l'application User XP : Les composants du

formulaire (zone de texte, liste déroulante...) sont accessibles à partir d'une palette. Les propriétés de chaque composant (nom, valeur, apparence, action...) sont exposées dans une fenêtre *Details*. Il est aisé d'établir le flux entre les formulaires. Enfin, des validateurs prédéfinis permettent de vérifier la validité des valeurs saisies par l'utilisateur (validateurs d'adresse de courriel, de valeurs numériques...).

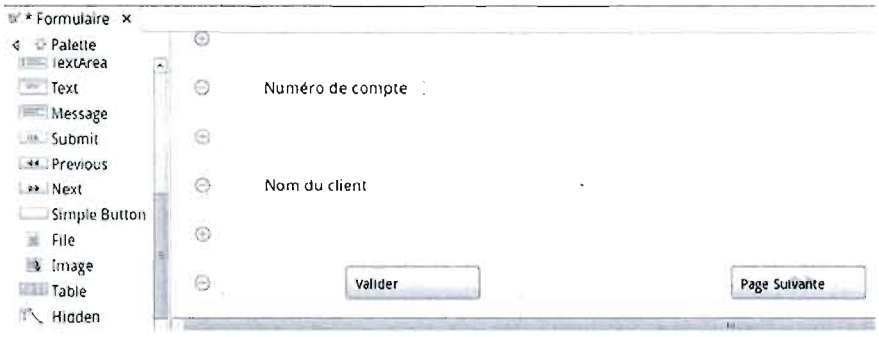


Figure 6.42 Éditeur de formulaires Bonita

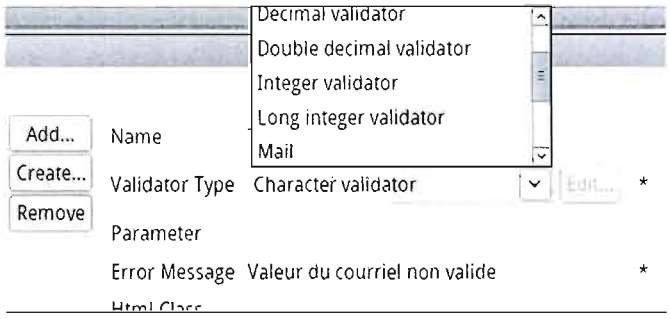


Figure 6.43 Validateurs de formulaires Bonita

Bonita répond amplement à ce critère en offrant des assistants puissants et bien conçus pour pratiquement toutes les fonctionnalités du système. Il obtient la note de 2 sur 2.

Enhydra Shark

L'onglet *Applications* de la fenêtre de propriétés d'un paquetage, d'un processus ou d'une activité permet de gérer les agents définis pour l'élément sélectionné.

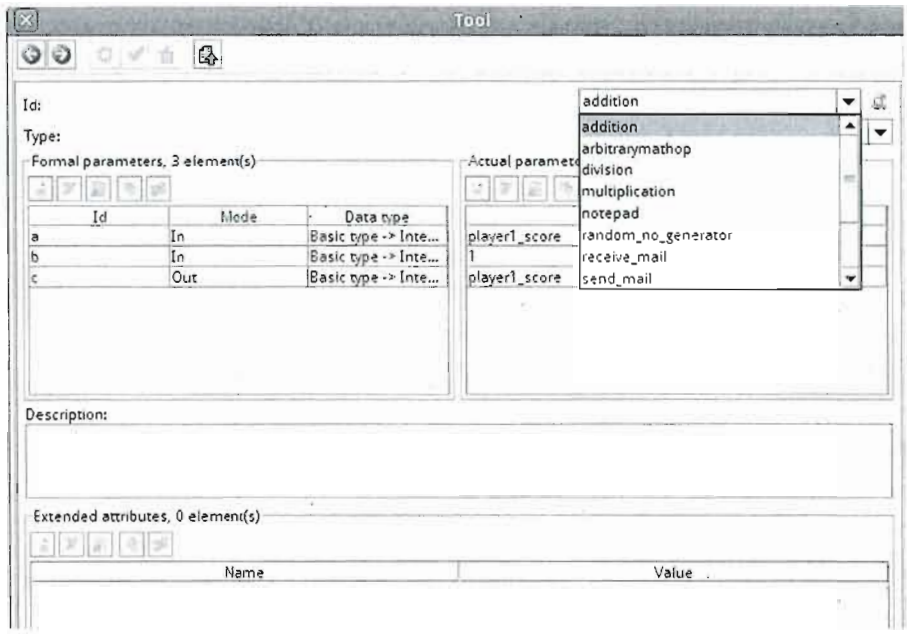


Figure 6.44 Assistant de définition d'un agent

Les propriétés des agents peuvent être éditées avec un assistant. L'outil gère les paramètres formels (identifiant, mode de lecture et type), leurs valeurs à l'invocation ainsi que d'éventuels attributs étendus passés de l'activité à l'agent (Figure 6.44).

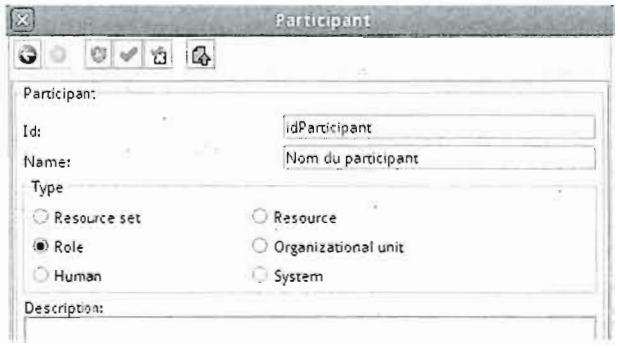


Figure 6.45 Assistant de définition des participants

Par ailleurs, il existe des assistants de gestion des participants (Figure 6.45). Les participants sont représentés graphiquement par des couloirs portant leurs noms. Toutes les

activités placées dans un couloir sont offertes au participant représenté.
Un assistant de définition de paramètres formels est représenté à la figure 6.46.

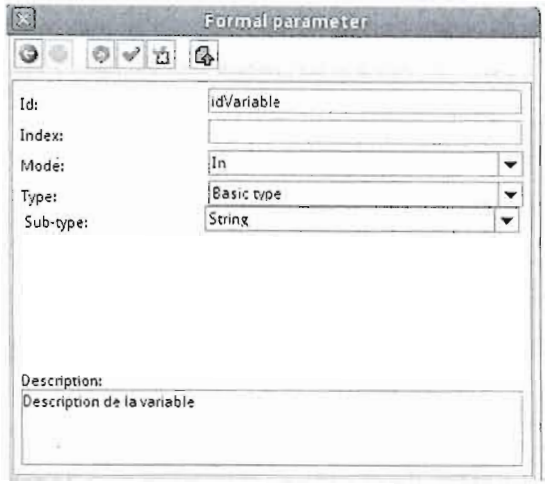


Figure 6.46 Assistant de définition d'un paramètre formel

Ainsi, *TWE* fournit des assistants pour administrer les données, les participants et les agents utilisés par les activités. Il obtient la note de 2 sur 2.

6.6.3 Interfaces accessibles web et optimisées pour les non informaticiens

jBPM

L'interface web Signavio permet de définir les flux de travail, d'en modifier les propriétés et d'en vérifier la validité. L'application organise les fichiers de flux dans des répertoires, exprimant une certaine structure logique au sein de l'organisme.
La console jBPM permet, quant à elle :

- de gérer les déploiements de définitions de processus et les unités de travail actives.
- d'afficher des rapports d'audit sur l'activité système, les instances de processus actives pendant une période donnée, etc.
- de parcourir les définitions de processus déployées, d'en créer de nouvelles instances et de supprimer des instances actives.

- d'afficher l'ensemble des tâches proposées à l'utilisateur connecté. L'utilisateur peut réclamer une tâche qui lui est proposée, démarrer et arrêter l'exécution d'une tâche qui lui est attribuée ou encore visualiser les propriétés de la tâche.

Figure 6.47 Gestion des processus dans la console jBPM

Les fonctions de gestion et de création d'instances de processus doivent être réservées aux administrateurs du système. La console jBPM permet d'implémenter ces contrôles d'accès grâce au système d'authentification.

Par conséquent jBPM offre des interfaces web accessibles aux utilisateurs métier. Ces interfaces sont intuitives et la console jBPM ajuste les droits d'accès selon le profil de l'utilisateur. Le produit obtient la note de 2 sur 2 sur ce critère.

Bonita Open Solution

L'interface utilisateur de l'application web User XP reprend la logique des boîtes de courriel. Bonita établit la correspondance suivante :

Interface User XP	Interface d'une boîte de courriel

Ouvrir une tâche en cours	Ouvrir un courriel reçu
Remplir les données du formulaire	Rédiger un courriel
Finir son action avec le bouton Soumettre	Envoyer un courriel
Assigner une tâche à une autre ressource	Réexpédier un courriel reçu à une autre personne

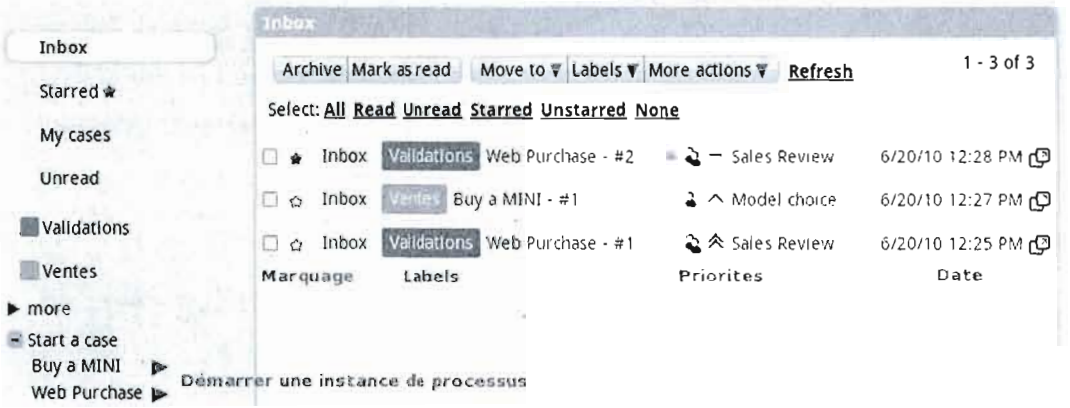


Figure 6.48 Boîte de réception pour une ressource

L'étoile à droite de chaque instance de processus permet de l'ajouter à la catégorie *Starred* (marquée). Il est possible d'assigner un ou plusieurs libellés aux instances de processus. Les libellés sont gérés en cliquant sur le lien *Manage Labels.*

Le bouton à droite du nom de l'instance de processus est vert si des tâches doivent être complétées, orange si une tâche est suspendue et gris lorsque toutes les tâches de l'instance de processus assignées à l'utilisateur connecté ont été exécutées.

Les priorités sont indiquées par un trait pour une priorité normale, un chevron pour une priorité élevée et deux chevrons pour une tâche urgente. Enfin, la date où la dernière tâche de l'instance de processus a été exécutée est indiquée.

Les actions de gestion des tâches (ré-assignation, modification de priorité...) sont directement accessibles à partir de l'interface utilisateur. Un utilisateur peut ajouter des

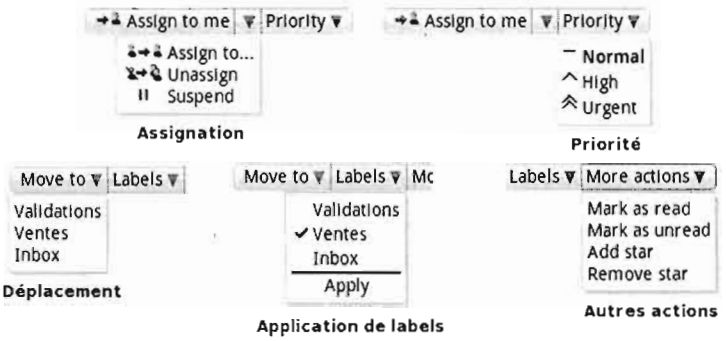


Figure 6.49 Actions possibles pour une tâche

commentaires aux différentes étapes des processus depuis User XP.

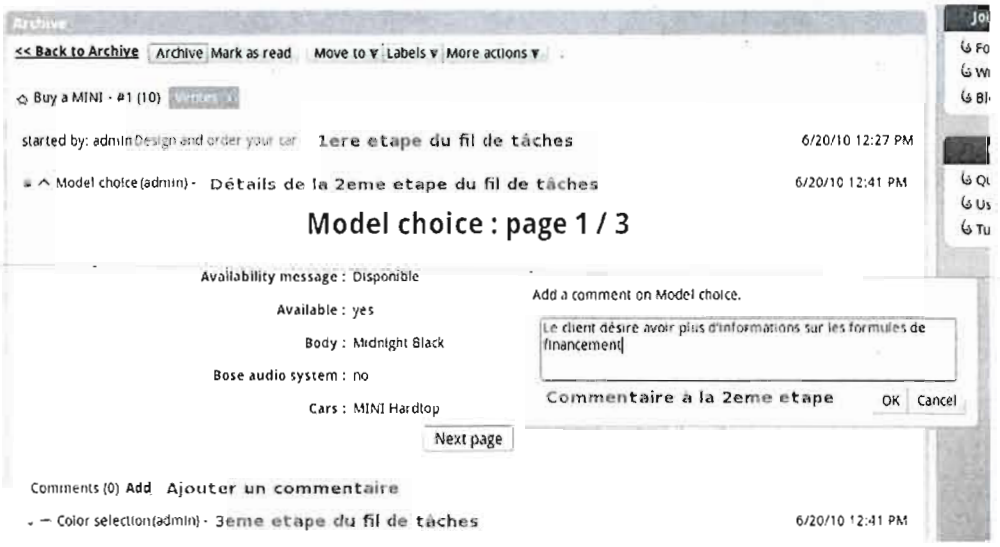


Figure 6.50 Fil des tâches d'une instance de processus

Lorsqu'une instance de processus est sélectionnée, l'historique des étapes précédentes est affichée, à la manière d'un fil d'échanges de courriels. Ceci est utile lorsqu'une décision dépend d'actions passées.

Les tâches d'administration sont restreintes aux utilisateurs ayant les droits correspondants. Un administrateur est capable de :

- Déployer et retirer des définitions de processus.

- Retirer toutes les instances d'un processus.
- Créer de nouvelles instances d'un processus.
- Gérer les instances de processus actives.
- Gérer les utilisateurs et les rôles.
- Afficher des statistiques générales (voir section *6.8.4 Journalisation et audit*).

L'interface User XP de Bonita permet aux utilisateurs de visualiser et d'exécuter leurs tâches. Les labels et les catégories les aident à structurer leurs items de travail. L'ensemble des actions possibles pour un utilisateur sont directement accessibles à partir de l'interface web. D'autre part, l'application instaure un système d'authentification et de gestion des permissions afin de contrôler l'accès aux fonctions d'administration du système. Le produit obtient une note de 2 sur 2.

Enhydra Shark

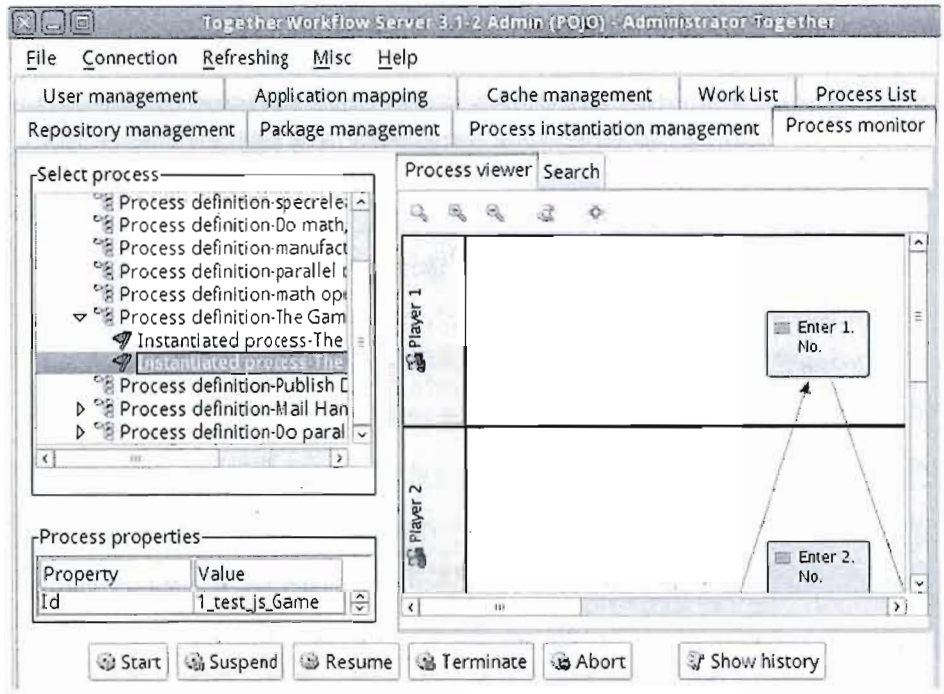


Figure 6.51 Version Swing de l'application d'administration

Enhydra Shark fournit une application d'administration déclinée en une version Swing

et une version Web. Les interfaces des deux versions sont identiques. Elles exposent neuf catégories de fonctionnalités d'administration, présentées chacune dans un onglet :

- Gestion des dépôts : Ajout et suppression de définitions de processus du dépôt. Un navigateur du système de fichiers permet de sélectionner les définitions XPDL à partir du disque local.
- Gestion des paquetages : Chargement de paquetages dans le moteur de flux depuis la liste de fichiers XPDL qui existent dans le dépôt. Mise à jour et retrait de paquetages.
- Gestion de l'instanciation des processus : Instanciation d'un processus appartenant aux paquetages chargés. Visualisation du diagramme et de la description d'un processus. Activation / Désactivation de l'instanciation de processus pour une définition donnée...
- Moniteur des processus instanciés : Démarrage, suspension, reprise et annulation d'instances. Consultation de l'historique, des variables et de la description de l'instance. Récupération des activités manuelles... Un bouton *Activity Management* permet de contrôler l'exécution du flux au niveau des activités de l'instance de processus sélectionnée.
- Gestion des utilisateurs : Ajout, suppression et modification des utilisateurs et des groupes. Correspondance entre les participants aux processus et les utilisateurs.
- Correspondances d'applications : Définition des correspondances entre les activités automatiques et les agents responsables de leur exécution.
- Gestion du cache : Administration de la taille de cache et du nombre de processus dans le cache.
- Liste de travail : Gestion de la liste de tâches offertes à une ressource (accepter, réallouer, déléguer, etc).
- Liste de processus : Liste des instances de processus démarrées par un utilisateur.

L'accès à l'application se fait en s'authentifiant avec un nom d'utilisateur et un mot de passe. Chaque utilisateur appartient à un ou à plusieurs groupes. Le fichier de configuration *SharkClient.conf* permet de modifier les permissions d'accès aux différentes catégories de fonctionnalités de l'application d'administration. Il est par exemple possible de restreindre les droits des utilisateurs non administrateurs à la simple gestion

de leurs propres listes de tâches. Ci-dessous, un extrait des paramètres pertinents du fichier *SharkClient.conf*, avec les valeurs par défaut :

```
RepositoryManagement.GroupPermission=admin
PackageManagement.GroupPermission=admin
ProcessInstantiationManagement.GroupPermission=admin
UserManagement.GroupPermission=admin
ProcesslistManagement.GroupPermission=admin,allusers
ProcesslistManagement.SelectUser.GroupPermission=admin
```

Ainsi, ES propose des interfaces destinées aux administrateurs et aux utilisateurs métier, accessibles comme applications de bureau ou à partir d'un navigateur web. Les applications utilisent un mécanisme d'authentification et permettent une gestion avancée des permissions accordées aux utilisateurs. Le produit obtient la note de 2 sur 2.

6.6.4 Abstraction des détails d'implémentation

jBPM

Les éditeurs graphiques de processus fournis par le plugiciel *eclipse* et l'application Signavio permettent d'abstraire le code XML sous-jacent. Toutefois, jBPM reste très étroitement lié au langage Java : la réalisation des patrons de flux nécessite très souvent de programmer avec du code Java. Très peu de fonctionnalités sont directement exprimables dans les modèles graphiques ou dans le langage XML. De plus, le code Java nécessaire à la réalisation de certains patrons est assez complexe et demande une bonne connaissance du langage.

Ainsi, la modélisation graphique est un moyen d'abstraire les détails d'implémentation, mais le recours à du code Java est souvent requis. Le produit obtient la note de 1 sur 2 pour ce critère.

Bonita Open Solution

Bonita fournit un effort important d'abstraction des détails techniques. En effet, la définition des processus se fait exclusivement via l'éditeur graphique et les multiples assistants de l'éditeur. De plus, les activités sont largement réalisables sans toucher à l'API Java de Bonita, et ce grâce aux multiples connecteurs.
Par ailleurs, la création des formulaires web de l'interface User XP bénéficie d'un éditeur graphique et d'assistants. Une application web complexe avec des validateurs et plusieurs formulaires pourrait ainsi être réalisée sans manipuler de code HTML ou JSP. Les développeurs et les designers web peuvent naturellement visualiser le code généré et effectuer des modifications avancées.
L'utilisateur n'a pas à maîtriser la création et le déploiement d'applications web puisqu'un bouton *Run* de l'éditeur Bonita crée l'archive, lance un conteneur web et y déploie l'application. Le navigateur web par défaut du système est lancé à l'adresse adéquate pour démarrer l'exécution de l'instance du processus déployé.
À partir de l'interface User XP, l'administrateur peut gérer les utilisateurs et les rôles, déployer et retirer des définitions de processus ou instancier des processus, sans manipuler l'API de Bonita.
Bonita abstrait donc les détails d'implémentation pour les utilisateurs métier. Il leur permet de concevoir les processus, les activités et les interfaces graphiques, de gérer les utilisateurs et leurs rôles, de déployer les applications web créées et de lancer leur exécution sans requérir de compétences en programmation. Le produit obtient une note de 2 sur 2.

Enhydra Shark

ES permet de définir les processus graphiquement grâce à l'éditeur *Together Workflow Editor*. Le code XPDL est automatiquement généré et il est consultable à partir de l'éditeur. Les définitions de processus réalisées avec des produits conformes à la norme XPDL peuvent être importées dans l'éditeur, qui génère automatiquement la

représentation graphique correspondante. Les éléments sont ajoutés graphiquement aux processus, en utilisant la palette de composants et les assistants de l'éditeur.
La gestion des activités, des participants et des données est également simplifiée.
Enfin, les applications d'administration permettent de gérer les processus, les activités, les utilisateurs et les listes de travail. Aucune connaissance de l'API n'est requise. Des opérations tel que le démarrage d'une activité, l'instanciation d'un processus ou l'ajout d'un groupe d'utilisateur sont faites simplement en appuyant sur des boutons, en cochant des cases ou en sélectionnant des entrées dans des listes déroulantes.
Le produit obtient une note de 2 sur 2 car il abstrait le code XPDL et Java sous-jacent.

6.6.5 Modèles de processus

jBPM

jBPM ne prend pas en charge les modèles de processus. La seule façon de réutiliser des modèles communs est de définir des flux génériques et d'en modifier des copies. Par exemple, un organisme de finance sociale et solidaire pourrait définir un processus "validation de demande d'emprunt" avec des noms de tâches génériques ou encore avec des parties incomplètes, destinées à être définies par les implémentations du modèles. Le processus de "validation d'une demande d'emprunt pour particuliers" pourrait partir d'une copie du processus générique et la modifier ou la compléter selon ses spécificités. jBPM obtient la note de 1 sur 2 sur ce critère car il ne gère pas directement les modèles de processus.

Bonita Open Solution

De manière similaire à jBPM, Bonita ne gère pas la notion de modèles de processus. Toutefois, on est capable de définir une version générique ou ayant des parties non complétées comme modèle de départ pour la définition d'autres processus. Le produit n'obtient que la note de 1 sur 2 sur cette fonctionnalité qu'il ne gère pas directement.

Enhydra Shark

TWE utilise le concept de *transient packages*. Ces paquetages sont toujours disponibles, indépendamment du processus en cours d'édition. L'utilisateur peut en copier des éléments (activités, participants...) dans ses propres processus.
Il est possible de modifier la liste des paquetages *transient* chargés au démarrage de l'éditeur avec le fichier de configuration *togwebasic.properties*. Par défaut, les implémentations ES des patrons de flux sont chargées.
On considère que ce concept correspond aux modèles de processus, car les éléments *transient* servent de modèles communs réutilisables dans les définitions de processus des utilisateurs. Une note de 2 sur 2 est accordée sur ce critère.

6.7 Le moteur de flux de travail

6.7.1 Intégration à l'environnement

jBPM

jBPM se base sur les langages Java et XML et fournit un plugiciel *eclipse* multiplateformes. Les applications de définition (Signavio) et de gestion des processus (console jBPM) sont accessibles à partir d'un navigateur web, s'exécutant sur n'importe quel système d'exploitation.
L'API Java du système est exhaustive et bien documentée. Elle permet de manipuler jBPM à partir d'applications tierces.
Un organisme est libre d'utiliser la console jBPM ou de la remplacer par une autre couche de présentation.
La console n'implémente pas le code d'interaction avec le moteur jBPM mais elle envoie des requêtes HTTP à un serveur REST nommé *gwt-console-server*. Le serveur est construit avec l'implémentation RESTeasy de la JSR 311 (spécification Java pour les services web REST). Il permet la publication rapide de ressources REST et prend en charge les formats de sérialisation "application/json" et "text/xml". D'autres formats

peuvent aisément être ajoutés. Ce choix architectural permet de découpler le client et le serveur et de communiquer avec plusieurs types d'interfaces clients.
Le code suivant est un exemple de requête au serveur REST pour récupérer les définitions de processus de l'utilisateur *utilisateur1* :

```
curl http://localhost:8080/gwt-console-server/rs/process/definitions -u
    utilisateur1:motdepasse
```

Par ailleurs, jBPM utilise le SGBD HyperSQL DB pour stocker les informations sur les processus, mais plusieurs scripts d'intégration permettent de recréer les schémas des tables jBPM dans d'autres SGBDs (Apache Derby, MySQL, PostgreSQL...).
Des scripts d'intégration existent également pour installer les applications web du système dans le conteneur Apache Tomcat et le serveur d'applications JBoss AS.
Les services de jBPM peuvent être utilisés avec le cadre de développement Spring. Le moteur de flux s'intègre également dans un environnement Java Entreprise. Les services web tiers peuvent être invoqués dans un flux de travail jBPM via du code Java.
En invoquant des services externes, jBPM se met en état d'attente, jusqu'à ce qu'il reçoive un signal externe pour poursuivre son exécution.
En résumé, jBPM se base sur des technologies ouvertes et adopte une architecture flexible et modulaire, ce qui lui permet de s'intégrer facilement à un environnement existant. Il obtient la note de 2 sur 2 sur ce critère.

Bonita Open Solution

L'éditeur Bonita est disponible pour les systèmes d'exploitation Windows, Mac et Linux. L'application web User XP est accessible à partir de tout système muni d'un navigateur web. Par ailleurs, la persistance des données se fait par défaut dans un SGBD H2. multi-plateformes. Enfin, l'API Bonita peut être utilisée dans tous les environnements munis de la machine virtuelle Java.
Les bibliothèques *bonita-server* permettent d'intégrer Bonita à une application Java. Bonita est également déployable sur un serveur d'applications, ce qui permet aux applications tierces de lui envoyer des requêtes à distance. Les descripteurs et les fichiers de

configuration pour le déploiement sur des serveurs JBoss, Jonas, WebLogic ou Glassfish sont prédéfinis.

L'exportation de la définition d'un processus génère les applications web de type standard, EJB 2 ou EJB 3 en fonction du serveur où l'utilisateur désire déployer les processus. L'environnement de déploiement est modifiable via la propriété *api-type* :

```
-Dorg.ow2.bonita.api-type=<Standard ou EJB2 ou EJB3>
```

Par ailleurs, les modules *BonitaAuth* et *BonitaStore* configurent l'authentification des utilisateurs. Le système de journalisation est interchangeable.

Concernant la couche de données, les fichiers *hibernate-core.properties* et *hibernate-history.properties* servent à configurer la source de données utilisée par le système. Hibernate est capable de s'interfacer avec la totalité des SGBDs populaires sur le marché.

Au niveau de la couche de présentation, l'utilisateur est capable de modifier l'apparence des interfaces générées et d'éditer le contenu des formulaires. Il peut également changer des propriétés avancées tel que le code HTML des formulaires. Les modèles définissant l'apparence globale de l'application sont configurables et il est possible de fournir ses propres modèles. Enfin, les modules de l'interface User XP peuvent être imbriqués individuellement dans des applications web existantes.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html dir="ltr" xml:lang="en"
xmlns="http://www.w3.org/1999/xhtml" lang="en">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF
            -8" />
        <meta name="description" content="Application de Formulaires
            Bonita" />
        <title>Mon application</title>
        <link href="css/bonita_form_default.css" rel="stylesheet "type=
            "text/CSS" />
    </head>
    <body>
            <element ID="bonita_form">
```

```
                </div>
        </body>
</html>
```

Enfin, les connecteurs Bonita permettent d'interagir avec une multitude d'applications externes. Si un connecteur retourne des résultats, Bonita est capable d'effectuer une correspondance entre les données reçues et les données des processus.
En définitive, Bonita est hautement intégrable aux environnements existants, à tous les niveaux. Le système est multi-plateformes et les couches données, métier et présentation sont totalement configurables. De plus, le système de connecteurs permet d'interagir avec une large palette d'applications externes. Le produit mérite une note de 2 sur 2.

Enhydra Shark

L'éditeur *Together Workflow Editor* n'est officiellement disponible que sous Windows, Linux et AIX. L'application étant codée en Java et son code source étant disponible, il est possible d'en générer un exécutable sous Mac. La variante web de l'application d'administration est accessible à partir d'un navigateur s'exécutant sur n'importe quel système d'exploitation. Par ailleurs, les bibliothèques de manipulation du moteur de flux sont écrites en Java et peuvent être utilisées dans tout système où un environnement de développement JDK de version supérieure ou égale à 1.4 est installé. Concernant le stockage des données, ES utilise par défaut une base de données HyperSQL DB, mais a été testé avec les SGBDs libres et multi-plateformes MySQL et PostgreSQL.
Le moteur de flux est accessible comme un ensemble d'objets Java "plats" (POJO - Plain Old Java Object) via des interfaces graphiques en Swing, Web ou même en ligne de commande. Il peut également être déployé comme EJB, comme service web ou comme objet distant accessible avec les protocoles RMI et CORBA.
Le système est largement configurable. Ses modules internes et son noyau peuvent être modifiés et remplacés par d'autres implémentations. De plus, les règles d'exécution du moteur de flux utilisent exclusivement des éléments de la spécification XPDL, sans faire intervenir d'extensions propriétaires sous forme d'attributs étendus. Le moteur peut

ainsi exécuter tout processus conforme à XPDL.

Par ailleurs, les agents peuvent interagir avec des services web ou encore recevoir des courriels. Il est possible de définir ses propres agents afin de gérer d'autres scénarios d'intégration. ES peut également importer les participants aux processus à partir d'une structure organisationnelle définie dans un dépôt LDAP.

Enfin, Enhydra DODS (Data Object Design Studio) est un outil de correspondance objet/relationnel développé par Enhydra. Il est utilisé pour persister les données. Via cet outil, le système peut être configuré pour utiliser toute sorte de SGBDs. Octopus est un autre module développé par Enhydra. Il s'agit d'un outil ETL (Extract Transform Load) dont se sert le système pour créer automatiquement les schémas de bases de données selon le type de SGBD choisi.

Ainsi, ES se base sur un langage de programmation répandu dans les systèmes informatiques d'entreprises (Java) et un langage de définition de flux standards (XPDL), ce qui facilite son intégration à des environnements existants. Les implémentations des couches de présentation et de données sont facilement modifiables. Le noyau même du moteur et ses modules internes sont configurables. Enfin, les agents et l'accès aux répertoires LDAP constituent des interfaces d'échange avec des systèmes externes. Le produit obtient une note de 2 sur 2.

6.7.2 Changements dynamiques

jBPM

Lorsqu'une nouvelle version d'un processus est déployée, les nouvelles instances du processus sont créées, par défaut, selon cette nouvelle définition. Il est possible de démarrer une nouvelle instance en spécifiant une ancienne version du processus. Toutefois, les instances déjà déployées gardaient leur version initiale jusqu'à l'introduction d'une nouvelle fonctionnalité dans jBPM 4.3, nommmée *instance migration.*

Cette fonctionnalité permet de migrer les instances de processus en cours d'exécution vers une nouvelle version. Pour cela, il suffit d'ajouter une balise <*migrate-instances*/>

dans la définition du processus.

```
<process name="nomprocessus">
  <start>
    <transition to="a"/>
  </start>
  <state name="a">
    <transition to="fin">
  </state>
  <end name="fin"/>
  <migrate-instances/>
</process>
```

Par défaut, lorsqu'une nouvelle version du processus *nomprocessus* est déployée, les instances démarrées avec la version précédente sont automatiquement mises à jour. *<migrate-instances action="end"/>* permet de terminer l'exécution des instances démarrées avec la version précédente, au lieu de les mettre à jour.
Pour mettre à jour les instances démarrées avec des versions autres que la version précédente, il suffit de spécifier les numéros de versions des instances à migrer. *<migrate-instances versions="2..4"/>* permet de mettre à jour les instances de versions 2, 3 et 4.
Il est possible de définir des expressions complexes pour les numéros de versions affectés par la mise à jour : *<migrate-instances versions="x-2..x"/>* indique les deux dernières versions et *<migrate-instances versions="*"/>* indique toutes les versions du processus.
Si les éléments de flux ont changé entre deux versions, on établit les correspondances comme ceci :

```
<migrate-instances>
    <activity-mapping old-name="b" new-name="a"/>
    <activity-mapping old-name="c" new-name="d"/>
</migrate-instances>
```

Pour gérer des mises à jour plus complexes, on crée une classe qui implémente l'interface *InstanceHandler* et sa méthode :

```
migrateInstance(ProcessDefinition nouvelleDefinition, ProcessInstance
    instanceAMigrer, MigrationDescriptor descripteurDeMigration)
```

La migration d'instances n'est actuellement définie que pour les éléments de flux de type état (*state*), mais il est prévu d'élargir la fonctionnalité aux autres éléments de flux. Ainsi, jBPM fournit une solution élégante et puissante pour gérer le changement dynamique des processus et obtient la note de 2 sur 2.

Bonita Open Solution

L'interface User XP permet de supprimer les anciennes versions d'un processus et d'en déployer de nouvelles sans arrêter le système. Toutefois, la migration des instances démarrées avec l'ancienne version vers la nouvelle n'est présentement pas possible avec Bonita. Il faudrait donc retirer toutes les instances actives d'une version donnée et en créer de nouvelles avec la version mise à jour. Les anciennes données sont ainsi perdues. La figure 6.52 montre la réalisation de cette opération à partir de l'interface User XP.

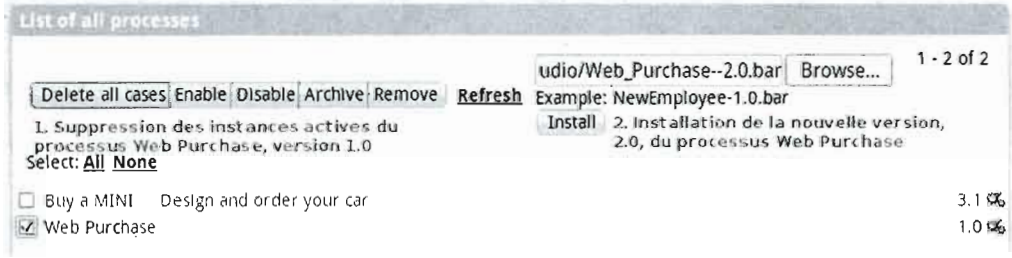


Figure 6.52 Déploiement à chaud d'un processus

Ainsi, Bonita est capable de déployer à chaud de nouvelles versions d'un processus mais ne gère pas la migration des instances créées avec l'ancienne version. Le produit obtient une note de 1 sur 2.

Enhydra Shark

Il est possible de charger de nouvelles définitions de processus pendant l'exécution du moteur de flux. La fonctionnalité est réalisable via les APIs d'ES ou avec les applications d'administration.

Les définitions de paquetages de processus chargées peuvent aussi être mises à jour. À partir de l'onglet de gestion des paquetages de l'application d'administration, on sélectionne la définition et on appuie sur le bouton *Update*. La liste des paquetages avec le même identifiant est affichée. En sélectionnant la version mise à jour, on s'assure que les prochaines instanciations de processus du paquetages se feront avec elle. Les processus déjà instanciés ne sont toutefois pas modifiés. Les fonctionnalités sont disponibles avec

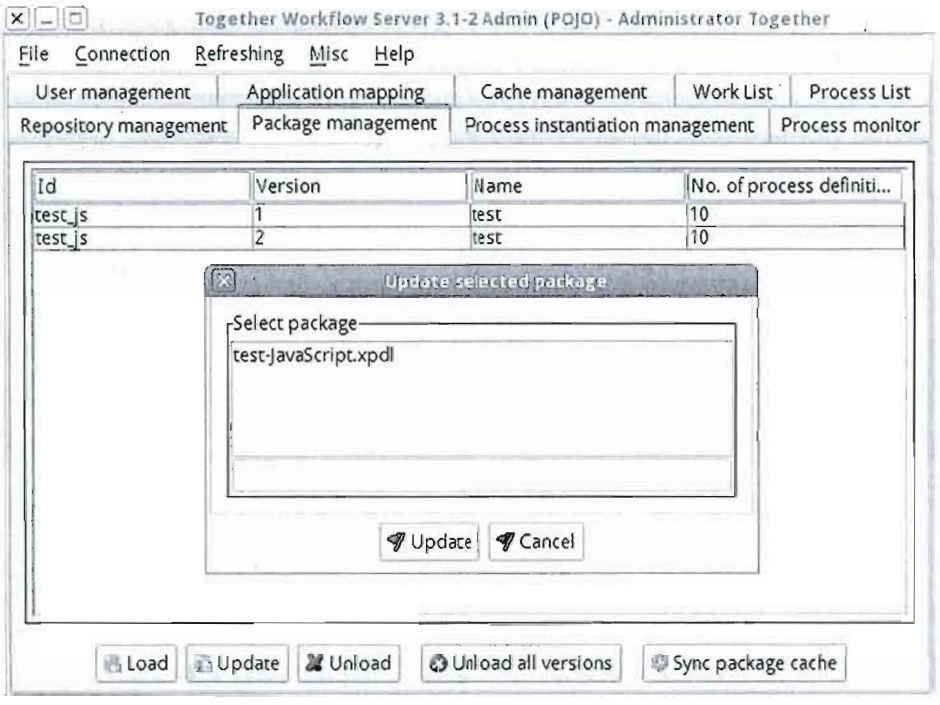


Figure 6.53 Mise à jour de la définition d'un processus

l'API :

```
updatePackage(WMSessionHandle shandle, String id, byte[] nouveauXpdl)
```

```
updatePackageFromFile(WMSessionHandle shandle, String id, String
    cheminFichier)
updateXPDL(WMSessionHandle shandle, String id, String version, byte[]
    nouveauXpdl, byte[] paquetageSerialise, long xpdlClassVer)
```

La première méthode met à jour le paquetage d'identifiant passé en paramètre. Le nouveau fichier XPDL est passé comme un tableau d'octets. La seconde méthode effectue la mise à jour en spécifiant le chemin du nouveau fichier.
Les applications d'administration exposent deux boutons *Unload* et *Unload all versions* qui permettent respectivement de retirer du moteur une version ou toutes les versions de paquetages ayants le même identifiant que l'élément sélectionné. L'opération nécessite qu'aucune instance de processus du paquetage ne soit actuellement active, et que le paquetage ne soit référencé par aucun processus actif. Un paquetage peut contenir un ou plusieurs processus.
Ainsi, ES permet de charger de nouveaux processus, de mettre à jour et de retirer d'anciennes versions pendant l'exécution du moteur de flux. Les instances déjà déployées ne sont toutefois pas migrées vers les nouvelles versions. Ce manquement fait que le produit n'obtient que 1 sur 2.

6.8 Outils de vérification

6.8.1 Vérification de la syntaxe

jBPM

Les flux de travail sont définis, aussi bien pour le langage jPDL que pour l'implémentation de BPMN, avec des fichiers XML. Le code personnalisé et l'API du système sont codés en Java. Ces deux langages (XML et Java) disposent d'une multitude d'éditeurs prenant en charge la vérification syntaxique. Entre autres, l'IDE *eclipse*, utilisé par le plugiciel GPD de jBPM, fournit la vérification syntaxique pour les deux langages.
Concernant les fichiers XML de définition de flux, l'ajout de la spécification jPDL ou BPMN dans l'en-tête des fichiers permet de bénéficier de l'auto-complétion et d'une

vérification syntaxique personnalisée. Le critère est donc satisfait, ce qui vaut une note de 2 sur 2.

Bonita Open Solution

Les processus sont modélisés graphiquement avec Bonita. Le code sous-jacent n'est donc pas exposé à l'utilisateur. L'éditeur vérifie la validité des diagrammes en temps réel et affiche un symbole d'erreur (cercle rouge) dans l'élément où une erreur s'est produite. En déplaçant le curseur sur le cercle, une fenêtre pop-up affiche les causes des erreurs. En plus des erreurs, l'éditeur affiche des avertissements lorsqu'un problème n'empêche pas l'exécution du flux de travail. Par exemple, la capture d'écran suivante montre une erreur pour une activité humaine qui n'est assignée à aucune ressource et un avertissement pour une étape qui ne possède pas de transition en entrée. En exportant le

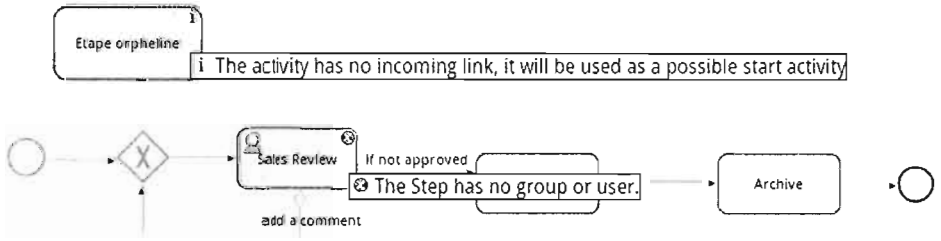


Figure 6.54 Vérification des diagrammes Bonita

diagramme sous format .proc, on obtient le code source de la définition des processus. Les fichiers .proc sont des fichiers XML obéissant à une syntaxe définie par Bonita. Le nom d'espace XML spécifié dans l'en-tête des fichiers .proc permet aux vérificateurs de syntaxe XML de valider leurs contenus. Étant donné que Bonita détecte les erreurs dans les diagrammes, cette approche n'est pas d'une utilité particulière.

Par ailleurs, Bonita est capable d'importer des définitions de processus sous format jBPM 3.2, BPMN 2.0 et XPDL 1.0. Ces trois formats se basent sur des balises XML et un nom d'espace XML. Par conséquent, ils peuvent bénéficier de la coloration syntaxique offerte par les éditeurs XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:form='
www.eclipse.org/gmf/runtime/1.0.2/notation" xmlns:process="http://www.bon
  <process:MainProcess xmi:id="_cDXC0XYQEd-HY6ttzlWDeA" name="Web_Purchas
bonitaModelVersion="5.2">
    <elements xmi:type="process:Pool" xmi:id="_cuRoEHYQEd-HY6ttzlWDeA" na
      <elements xmi:type="process:Task" xmi:id="_c-ZHsHYQEd-HY6ttzlWDeA" n
```

Figure 6.55 Coloration syntaxique du code XML généré à partir d'un diagramme Bonita

Enfin, les éditeurs Groovy et Java disponibles dans Bonita utilisent la coloration syntaxique fournie par la plate-forme *eclipse*, sur laquelle est bâti l'éditeur Bonita.
Bonita permet donc la vérification en temps réel de la validité des diagrammes. Quant aux formats XML pris en charge par le système, ils peuvent utiliser la coloration syntaxique offerte par les éditeurs XML. Le produit obtient une note de 2 sur 2.

Enhydra Shark

L'éditeur *TWE* dispose d'un vérificateur de validité des définitions de processus qui peut être exécuté en appuyant sur l'icône correspondante ou en sélectionnant l'entrée *Package — Check Validity* de la barre de menus. S'il existe des erreurs ou des avertissements, la vue *Problems* affiche leur type (problème de connexion, de logique, de conformité ou de schéma), leur description, l'élément où elles surviennent ainsi que le paquetage et le processus parent. Les entrées de la liste de problèmes peuvent être triées selon l'une de ces informations.

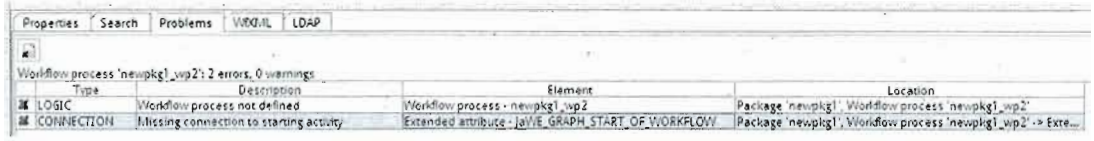


Figure 6.56 Liste des problèmes de validation

Cliquer sur les cellules *Element* et *Location* d'une entrée de la liste de problèmes sélectionne respectivement l'élément dans le diagramme et le paquetage auquel il appartient dans la vue *Navigator*. Double-cliquer sur une entrée ouvre l'assistant d'édition

de ses propriétés.

Le fichier de configuration *togwecontroller.property* permet de spécifier si les erreurs sont automatiquement vérifiées pendant la conception ou si elles ne le sont que sur demande de l'utilisateur.

Si on préfère manipuler du code XPDL plutôt que des diagrammes, les éditeurs XML ou XPDL (éventuellement fournis par d'autres systèmes de gestion de flux) peuvent utiliser le nom d'espace XML du format pour réaliser la détection d'erreurs syntaxiques. D'ailleurs, la vue XPDL de *TWE* dispose de la coloration syntaxique.

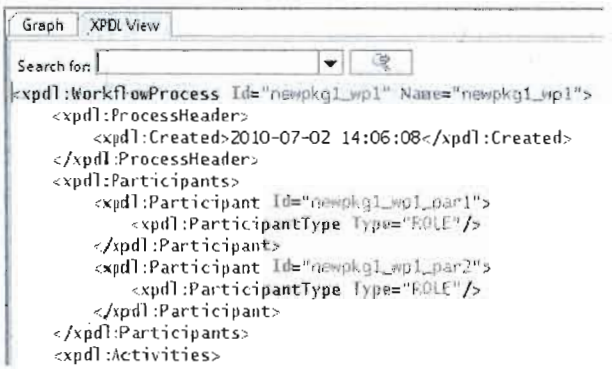


Figure 6.57 Coloration syntaxique avec la vue XPDL de TWE

Les fonctionnalités de validation sont également accessibles à partir de l'API. Les méthodes *getXPDLValidationErrors()* des classes *PackageInvalid* et *ExternalPackageInvalid* retournent un texte formaté en HTML qui liste toutes les erreurs de validation survenues lors du chargement ou de la mise à jour de paquetages.

Ainsi, l'éditeur *TWE* valide les processus en temps réel ou à la demande. Sa vue XPDL offre la coloration syntaxique. Des méthodes de l'API d'ES permettent de valider des paquetages de processus. Enfin, des éditeurs XML ou XPDL tiers peuvent utiliser la spécification du format XPDL pour vérifier la syntaxe des fichiers. Une note de 2 sur 2 est attribuée sur ce critère.

6.8.2 Test

jBPM

L'API de jBPM étant en Java, il est possible d'utiliser n'importe quelle bibliothèque de test disponible pour le langage (JUnit, TestNG...).
De plus, des classes de test étendant les tests unitaires JUnit sont fournies par le système. La classe *JbpmTestCase* expose plusieurs méthodes utilitaires pour tester l'exécution des processus. Les services du moteur de flux sont initialisés par la classe de test et disponibles comme variables d'instance. Par exemple, les attributs *executionService*, *taskService* et *historyService* référencent les services de gestion de l'instance en cours d'exécution, des tâches et de l'historique d'exécution.
Les méthodes utilitaires tel que *assertActivityActive* et *assertProcessInstanceEnded* facilitent la création des tests. La méthode *tearDown*, qui s'exécute après chaque test, est redéfinie dans *JbpmTestCase* pour s'assurer que toutes les tables de la base de données sont vides.
Le code suivant utilise la variable *repositoryService* pour déployer et supprimer les fichiers de processus. La variable *executionService* est utilisée pour démarrer une instance de processus :

```
public class ExempleDeTest extends JbpmTestCase {
  String deploymentId;

  protected void setUp() throws Exception {
    deploymentId = repositoryService.createDeployment()
        .addResourceFromClasspath("org/nosprocessus/unprocessus.jpdl.xml")
            .deploy();
  }

  protected void tearDown() throws Exception {
    repositoryService.deleteDeploymentCascade(deploymentId);
    super.tearDown();
  }
```

```
  public void testUneFonctionnaliteDuProcessus() {
    ProcessInstance processInstance = executionService.
        startProcessInstanceByKey("NomDuProcessus");
    //suite du test...
  }
}
```

En plus de *JbpmTestCase*, les classes *JbpmEjbMockTestCase* et *JbpmJMSMockTestCase* permettent d'utiliser respectivement des EJB et des queues JMS factices à des fins de test.

La classe *AbstractTransactionalJbpmTestCase* permet de tester un processus en isolation, sans toucher à la base de données. Elle étend la classe *AbstractTransactionalDataSourceSpringContextTests* du cadre Spring. Ceci permet de tester les processus selon une approche similaire au test d'objets d'accès aux données (DAO) avec Spring.

Ainsi, jBPM bénéficie des bibliothèques de test du langage Java et fournit ses propres classes de tests. Il obtient la note de 2 sur 2.

Bonita Open Solution

Via la modélisation graphique et les multiples assistants, Bonita abstrait le code XML et Java utilisé dans les processus. Toutefois, l'API du système étant codée en Java, il est possible d'utiliser les bibliothèques de test disponibles pour le langage. On est alors capables de définir des tests unitaires et des suites de tests où les processus sont déployés localement (à l'aide de la bibliothèque *bonita-server*).

Par ailleurs, l'éditeur Bonita permet de tester le fonctionnement des connecteurs. En sélectionnant *Connectors* $->$ *Test a connector* dans le menu principal, on teste le fonctionnement d'un connecteur en fournissant des paramètres en entrée. En appuyant sur *Evaluate*, Bonita lance le test et affiche soit les résultats, soit une erreur avec sa trace complète.

De plus, l'éditeur Groovy peut affecter des valeurs de test aux variables de processus et évaluer le résultat produit par l'expression Groovy saisie dans l'éditeur.

Figure 6.58 Test du connecteur Drools

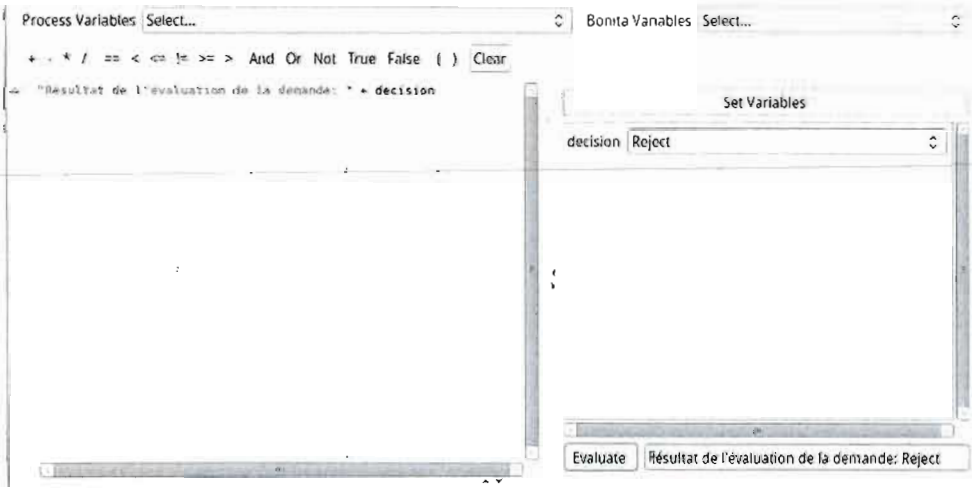


Figure 6.59 Test des expressions Groovy

Enfin, l'éditeur de formulaires User XP permet de prévisualiser l'apparence des formulaires avant leur déploiement.

Ainsi, l'API de Bonita peut être utilisée avec les bibliothèques de test Java. Les connecteurs, les expressions Groovy et les formulaires User XP peuvent être testés en dehors de l'environnement de production. Le produit obtient la note de 2 sur 2 sur ce critère.

Enhydra Shark

Enhydra Shark étant utilisable comme bibliothèque dans une application Java, il peut bénéficier des outils de test disponibles pour le langage. Les tests unitaires sont facilement réalisables du fait qu'ES peut être instancié comme POJO, sans avoir à le déployer sur un serveur d'applications.

Le code suivant est un test unitaire JUnit qui récupère une instance d'ES à l'initialisation (méthode *setUp*). La méthode *testListeTravailNonVide* crée une tâche et vérifie que la liste de travail est non vide. La méthode *textExistenceProcessusEnExecution()* instancie un processus et vérifie qu'il existe des processus en cours d'exécution.

```
import junit.framework.TestCase;
...
public class ExempleSharkTest extends TestCase {

    UserTransaction ut = null;

    protected void setUp() throws Exception {
        LocalContextFactory.setup("sharkdb");
        ut = (UserTransaction) new InitialContext().lookup("java:comp/
            UserTransaction");
        ...
        Shark.configure("/opt/tws-3.1/conf/Shark.conf");
        Shark shark = Shark.getInstance();
        ut.commit();
    }

    public void testListeTravailNonVide() throws Exception {
        ut.begin();
        ...
        int nbElements = sc.getResourceObject().how_many_work_item();
        assertTrue("La liste de travail ne doit pas etre vide", nbItems >
            0);
    }
```

```
    public void testExistenceProcessusEnExecution() throws Exception {
        ...
        ProcessFilterBuilder fb = Shark.getInstance().
            getProcessFilterBuilder();
        WMFilter f = fb.addStateEquals(shandle, "open.running");
        String query = fb.toIteratorExpression(shandle, f);
        wpi.set_query_expression(query);
        WfProcess[] procs = wpi.get_next_n_sequence(0);
        ut.commit();

        assertNotNull("Il doit exister des instances de processus en
            execution", procs);
    }
}
```

Étant donné que l'API d'ES ne fournit pas ses propres classes de test et que l'éditeur *TWE* n'incorpore pas d'outils de test, le produit n'obtient que la note de 1 sur 2 pour ce critère.

6.8.3 Débogage

jBPM

Les IDEs Java fournissent des outils de débogage évolués. Au niveau des flux de travail, la console jBPM peut afficher l'état d'un processus en cours d'exécution en indiquant l'étape actuellement active avec un curseur (triangle rouge). La console affiche aussi les valeurs des variables de processus pendant l'exécution.
jBPM utilise le cadre Hibernate pour stocker les données du système. On est capable de déboguer la procédure de stockage en configurant la journalisation de Hibernate au niveau de détail le plus élevé :

```
org.hibernate.SQL.level=FINEST
org.hibernate.type.level=FINEST
```

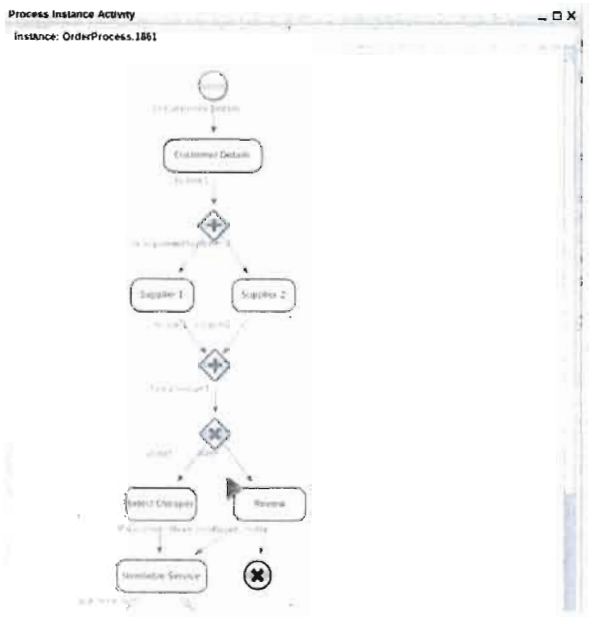


Figure 6.60 Suivi de l'exécution du processus

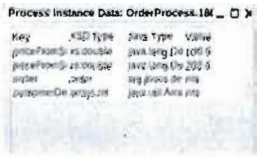


Figure 6.61 Affichage des valeurs des variables pendant l'exécution

Les propriétés suivantes permettent de rapporter plus d'informations dans les journaux relatifs à Hibernate :

```
hibernate.show_sql = true
hibernate.format_sql = true
hibernate.use_sql_comments = true
```

Ainsi, jBPM utilise les fonctionnalités de débogage du code Java et de la persistance Hibernate. Il offre ses propres outils pour déboguer les flux de travail. Le produit obtient une note de 2 sur 2.

Bonita Open Solution

Les IDEs Java fournissent des outils de débogage évolués. De plus, ils peuvent déboguer l'exécution de l'application User XP. En utilisant *eclipse*, on sélectionne l'archive *war* du processus à déboguer et on établit une configuration de débogage de type "application Java distante". De cette manière, on exécute l'application User XP en mode hébergé (hosted).

Par ailleurs, Bonita utilise le cadre Hibernate pour persister les données. Il bénéficie donc de la journalisation détaillée de l'outil.

L'éditeur graphique Bonita ne permet pas de fixer des points d'arrêts à certaines étapes des processus. On pourrait cependant créer des connecteurs qui suspendent l'exécution du flux de travail et affichent les valeurs des variables de processus. À cet effet, la communauté Bonita a produit un connecteur nommé Logger [16] qui permet de journaliser les données de flux. Le connecteur prend en paramètre le nom de l'étape qui génère les entrées dans le journal ainsi que des tuples nom, valeur des données à journaliser.

Ainsi, Bonita ne propose pas d'outils de débogage mais il est possible d'utiliser ceux des IDEs Java. Un connecteur fourni pas la communauté permet de tracer les valeurs des données pendant l'exécution des processus. Le produit obtient la note de 1 sur 2 pour ce critère car les outils de débogage proviennent d'autres applications.

Enhydra Shark

Ni l'éditeur *TWE* ni les applications d'administration ne permettent de définir des points d'arrêt. Il est toutefois possible d'inspecter la valeur des variables de processus pendant l'exécution via la fonction *Update Variables* qui affiche la valeur actuelle des variables. De plus, l'onglet *Search* du moniteur de processus permet d'effectuer des requêtes selon l'état actuel des processus, activités et variables. Par exemple, la figure 6.62 affiche les résultats de la recherche des processus où une variable *counter* a une valeur supérieure

16. http ://www.bonitasoft.org/exchange/extension_view.php?eid=23 (dernière consultation le 09-11-10)

à zéro. On note que les deux processus satisfaisant la requête sont en état d'exécution (open.running). Ces fonctionnalités sont disponibles via l'API d'ES.

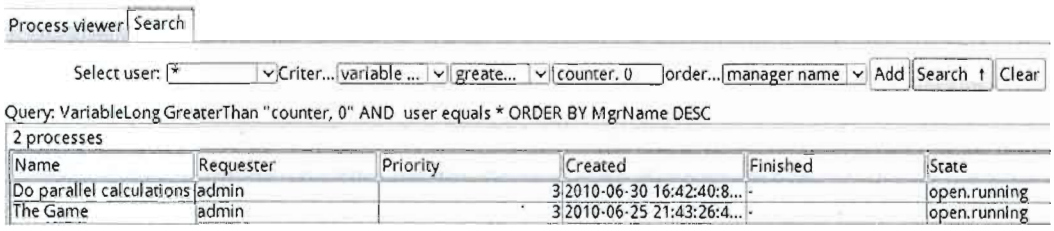


Figure 6.62 Requêtes de sélection pendant l'exécution de processus

Les points d'arrêts peuvent être définis dans un IDE, lorsqu'on manipule Enhydra via son API Java.

Enfin, la journalisation est personnalisable (voir critère *6.8.4 Journalisation et audit*). Des journaux détaillées peuvent constituer un outil précieux pour le débogage.

Ainsi, ES permet d'inspecter les processus et leurs propriétés à l'exécution. Pour les applications qui manipulent le produit via son API, il est possible de définir des points d'arrêts et de profiter des outils de débogage fournis par les IDEs. Enfin, la journalisation fournie par ES peut constituer une aide au débogage. Le produit obtient la note de 2 sur 2.

6.8.4 Journalisation et audit

jBPM

Les opérations du moteur de flux sont journalisées avec une extension jBPM des bibliothèques de journalisation Java ou avec un service d'historique :

- La classe *LogFormatter* fait partie de l'API de la machine virtuelle de processus (PVM). Elle permet de formater l'affichage des informations journalisées. Le fichier *logging.properties* configure la journalisation :

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINEST
```

```
java.util.logging.ConsoleHandler.formatter = org.jbpm.pvm.internal.log.
    LogFormatter
# On indique que seuls les messages de niveau SEVERE relatifs au
    paquetage org.banque.retraits
# sont journalises:
org.banque.retraits.level = SEVERE
org.jbpm.level=FINE
org.jbpm.tx.level=FINE
```

- jBPM maintient un historique d'exécution dans une base de données. L'information est mise à jour en temps réel dans le cadre de la transaction d'exécution du processus ou de la tâche. L'historique persiste après la fin de l'exécution. Les événements d'historique modifient des entités existantes ou en créent de nouvelles. Par exemple, un événement *ProcessInstanceCreate* va créer une entité de type *HistoryProcessInstance*, tandis qu'un événement *ProcessInstanceEnd* va modifier la propriété *endTime* de l'entité *HistoryProcessInstance* existante.

  Le service couvre quatre types d'entités : les instances de processus, les instances d'activités, les tâches et les variables. Chaque entité possède sa propre liste de propriétés journalisées.

  L'accès à l'historique se fait via le service *HistoryService* de l'API de jBPM.

Par ailleurs, la console jBPM permet de produire des rapports d'audit sur l'activité système et les instances de processus actives pendant une période donnée.
Ainsi, en plus des outils de journalisation de Java et du cadre Hibernate, jBPM fournit ses propres bibliothèques de journalisation et d'audit. Le système obtient la note de 2 sur 2 sur ce critère.

Bonita Open Solution

Bonita utilise les APIs de journalisation standards de Java[17]. Plusieurs paramètres permettent d'ajuster les journaux produits. Certains paramètres sont relatifs à la jour-

17. (http ://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/LogManager.html) (dernière consultation le 09-11-10)

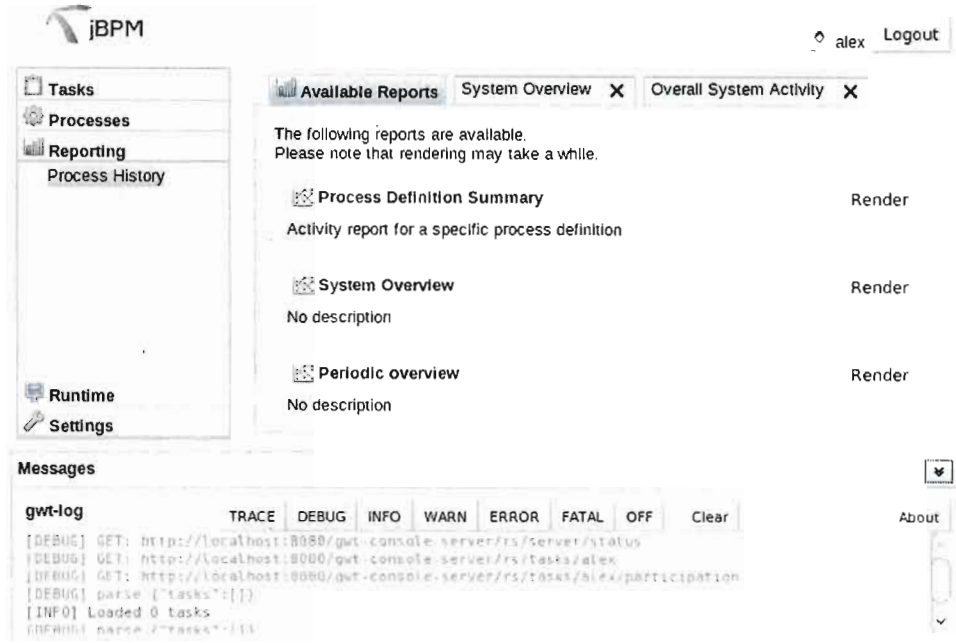


Figure 6.63 Génération de rapports dans la console jBPM

nalisation Java en général :

```
java.util.logging.FileHandler.pattern = %t/bonita.log
java.util.logging.FileHandler.formatter = java.util.logging.
    SimpleFormatter
java.util.logging.FileHandler.level = ALL
...
```

D'autres propriétés sont spécifiques à Bonita :

```
#org.ow2.bonita.level = ALL
org.ow2.bonita.example.level = FINE
#org.ow2.bonita.pvm.level = WARNING
org.bonitasoft.level = ALL
...
```

D'autres encore concernent la persistance des données :

```
org.hibernate.cache.EhCacheProvider.level = SEVERE
```

```
org.hibernate.cfg.Environment.level = INFO
org.hibernate.engine.StatefulPersistenceContext.level = SEVERE
...
```

Les niveaux OFF, SEVERE, FINE, WARNING, INFO, ALL établissent le niveau de détail des journaux. Comme le montrent les extraits du fichier de configuration précédents, on peut établir des niveaux différents pour chaque aspect de l'exécution.
En plus des journaux Java, Bonita génère des journaux spécifiques au moteur d'exécution et les enregistre dans un répertoire *logs* sous la racine du répertoire d'installation de Bonita. La dénomination des journaux suit le format :

```
ReportEngine\_AAAA\_MM\_JJ\_HH\_MM\_SS.log
```

Bonita crée une base de données de l'historique d'exécution des processus, en parallèle aux journaux Hibernate par défaut. La persistance Hibernate par défaut et la base de données de l'historique disposent chacune de leur propre fichier de configuration :

```
<hibernate-configuration name='hibernate-configuration:core'>
<properties resource='hibernate-core.properties' />
...
<hibernate-configuration name='hibernate-configuration:history'>
<properties resource='hibernate-history.properties' />
...
```

Par ailleurs, lorsque les processus sont déployés dans des conteneurs web ou des serveurs d'applications, ces derniers possèdent leurs propres fichiers de journalisation, relatifs au déploiement et aux éventuelles erreurs d'exécution des requêtes client.
Concernant l'audit, l'interface User XP génère des rapports sur le nombre d'instances de processus actives, les nombres de processus dont l'exécution a été complétée, les tâches actives, le pourcentage de tâches urgentes/de priorité élevée et normale, etc. Les données sont triées par journée de travail et représentées avec des diagrammes clairs et expressifs.
Ainsi, Bonita permet de configurer les différents aspects de journalisation, fournit des journaux dédiés au moteur de flux, persiste les historiques d'exécution dans une base de

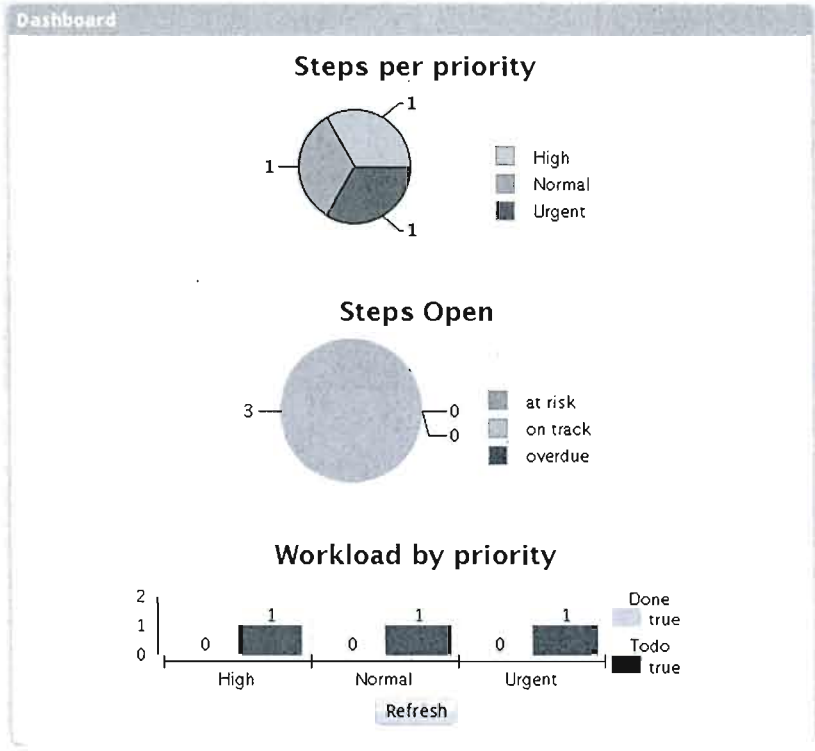


Figure 6.64 Rapport d'audit User XP

données et propose des rapports d'audit via l'application User XP. Le produit répond aux exigences et obtient une note de 2 sur 2.

Enhydra Shark

Shark fournit une implémentation par défaut de la journalisation qui utilise Apache log4j [18] Il est possible de modifier l'outil de journalisation car ES utilise une façade *org.enhydra.shark.api.internal.logging.LoggingManager* qui le découple du sous-système de journalisation utilisé.

L'API de journalisation en entier peut être remplacée. La classe de gestion de journalisation est spécifiée par le paramètre de configuration *LoggingManagerClassName*. La

18. http ://logging.apache.org/log4j/ (dernière consultation le 09-11-10)

journalisation est désactivée en supprimant le paramètre. Les niveaux de journalisation sont également configurables.

La configuration par défaut génère des informations sur :

- Le flux d'exécution du serveur : chargement de paquetages, instanciation de processus, complétion d'activités...
- Les chargements, retraits et mises à jour de paquetages.
- La persistance des données : communications avec les bases de données.

ES bénéficie de la génération de journaux sous format XML offerte par log4j.

Concernant l'audit, Enhydra implémente l'interface 5 de la spécification WfMC. Cette interface définit le format des données d'audit que doit générer un système de gestion de flux. Par défaut, les données d'audits sont archivées pour éventuellement générer des statistiques à long terme.

Enhydra Shark se sert de log4j comme outil de journalisation mais il est possible d'utiliser d'autres solutions. Les paramètres de journalisation sont hautement configurables.

Par ailleurs, ES fournit une implémentation de la fonctionnalité d'audit conforme à l'interface 5 de la spécification WfMC. Une note de 2 sur 2 est accordée sur ce critère.

6.9 Stockage des processus

6.9.1 Versionnage

jBPM

Lorsqu'un processus est déployé, un identifiant lui est automatiquement assigné. L'identifiant est généré selon le format *clé-version*.

La clé est créée par le système à partir du nom du processus, en remplaçant les caractères non alpha-numériques par des tirets-bas. Un nom de processus est associé à une clé unique et vice-versa.

Si le numéro de version du processus à déployer n'est pas spécifié, le système en assigne un. Sa valeur sera le plus grand numéro de version des définitions de processus déployées avec la même clé, incrémenté de un. Si aucune définition de processus avec la même clé

n'existe, le numéro assigné est 1.

Lorsqu'on démarre une instance de processus en spécifiant uniquement une clé, la méthode de service choisit la version la plus récente du processus portant cette clé pour créer la nouvelle instance.

```
ProcessInstance processInstance = executionService.
    startProcessInstanceByKey("ICL");
```

Lorsqu'une nouvelle version du processus est déployée, toutes les instances créées ultérieurement le seront à partir de cette dernière version.

La méthode *startProcessInstanceById* prend en paramètre la combinaison clé-version du processus à instancier.

```
ProcessInstance processInstance = executionService.
    startProcessInstanceById("ICL-1");
```

Ainsi, jBPM gère automatiquement le versionnage des processus. Il permet à l'utilisateur de démarrer des instances en spécifiant un numéro de version. Une note de 2 sur 2 est accordée sur ce critère.

Bonita Open Solution

Bonita permet de spécifier un numéro de version pour un diagramme et pour chaque couloir (pool) du diagramme. Un couloir est sensé contenir la définition d'un seul processus. De manière similaire à jBPM, les identifiants des processus Bonita sont composés de leur nom et de leur numéro de version. Lorsqu'un processus est exporté, le nom par défaut proposé par Bonita est constitué de la combinaison nom du processus–numéro de version.

Une note de 2 sur 2 est accordée sur ce critère.

Enhydra Shark

La section *redefinable header* des définitions de paquetages et de processus englobe les attributs qui peuvent être spécifiés pour un paquetage et redéfinis pour chaque processus

Figure 6.65 Versionnage Bonita

du paquetage.

Les informations relatives au versionnages sont :

- La version du paquetage ou du processus.
- L'état de publication : Les valeurs possibles sont *UNDER_REVISION*, *RELEASED* et *UNDER_TEST*. Si la valeur est *RELEASED*, l'éditeur *TWE* ne permet pas de sauvegarder le document lorsqu'il contient des erreurs ou des avertissements. Ce comportement par défaut peut être modifié via la propriété *AllowInvalidPackageSaving* du fichier de configuration *togwecontroller.properties*.

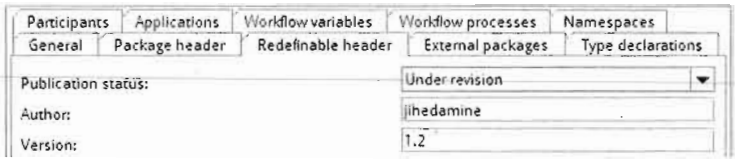


Figure 6.66 Versionnage des paquetages et des processus ES

Les changements sont répercutés dans le code XPDL :

```
<xpdl:RedefinableHeader PublicationStatus="UNDER_REVISION">
  <xpdl:Author>jihedamine</xpdl:Author>
  <xpdl:Version>1.2</xpdl:Version>
</xpdl:RedefinableHeader>
```

Par ailleurs, l'application d'administration définit des versions de déploiement. Lorsqu'un paquetage est chargé pour la première fois, il porte la version numéro 1. S'il est mis à jour, la nouvelle version porte le numéro 2 et ainsi de suité. Cette stratégie permet de retirer du moteur de flux une version spécifique ou toutes les anciennes versions des définitions du paquetage.

ES permet donc de versionner les définitions de processus et de paquetages ainsi que les chargements de définitions dans le moteur de flux. Une note de 2 sur 2 est attribuée sur ce critère.

6.9.2 Importation/Exportation de processus

jBPM

jBPM permet uniquement de déployer des processus sous format .jpdl.xml ou .bpmn.xml, mais ne propose pas de fonctionnalités d'importation ou d'exportation de processus vers des formats autres que ceux définis par le système. D'anciennes versions de jBPM proposaient cette fonctionnalité.
Le projet d'implémentation de la norme BPMN en est encore à ses débuts. Plusieurs fonctionnalités pouvant être modélisées avec la norme ne sont pas implémentées dans jBPM.
L'importation et l'exportation de processus n'est pas prise en charge par le produit, qui obtient une note de 0.

Bonita Open Solution

En plus de son format propriétaire .proc, Bonita est capable d'importer des définitions de processus sous format XPDL 1.0, BPMN 2.0 et jBPM 3.2. Les processus importés sont convertis en processus Bonita et le diagramme correspondant est automatiquement généré.
En revanche, le format .proc de Bonita est le seul format d'exportation possible.
Le produit obtient la note de 1 sur 2 car il est capable d'importer des processus définis dans les formats tiers les plus populaires mais ne permet d'exporter que dans son format propriétaire. Il s'agit probablement d'un choix visant à amener les utilisateurs à utiliser Bonita plutôt qu'une autre solution.

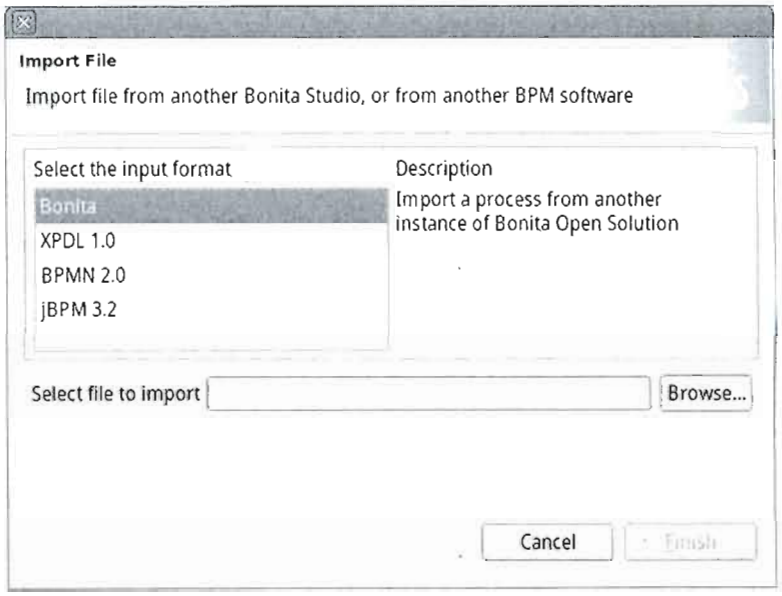


Figure 6.67 Importation de définitions de processus dans Bonita

Enhydra Shark

Le moteur d'Enhydra Shark, est capable de charger et d'exécuter toute définition de processus conforme au standard XPDL. En effet, les règles d'exécution du moteur utilisent exclusivement des éléments de la spécification XPDL, sans faire intervenir d'extensions propriétaires sous forme d'attributs étendus. L'éditeur *TWE* permet également d'éditer tout fichier XPDL. La représentation graphique est automatiquement générée par l'éditeur.

Les processus créés avec ES sont enregistrés sous format XPDL. Ils peuvent donc être exportés vers d'autres moteurs de flux conformes à la norme.

En adoptant le standard XPDL, ES permet l'interopérabilité avec d'autres moteurs de flux conformes à la norme. Ceci permet l'importation et l'exportation de processus entre des produits compatibles. Une note de 2 sur 2 est attribuée sur ce critère.

6.10 Conformité aux standards

jBPM

Le langage de description de flux de jBPM est jPDL. Même si le langage est à code ouvert et sous licence libre, il ne constitue pas un standard adopté pour la modélisation des processus.
La spécification BPMN définit un langage riche, capable d'exprimer des constructions de flux de travail avancées mais seuls quelques éléments basiques sont présentement implémentés sur la machine virtuelle de processus de jBPM. De plus, les éditeurs graphiques du plugiciel *eclipse* et de l'application Signavio ne prennent en charge que la modélisation des processus jPDL.
Une ancienne version de jBPM, nommée jBPM BPEL prenait en charge le standard BPEL. Le projet a depuis été avorté.
Une prise en charge partielle pour BPMN et inexistante pour BPEL et XPDL valent au produit une note de 1 sur 2.

Bonita Open Solution

Les diagrammes de définition de processus Bonita sont conformes à la norme de modélisation BPMN 2.0. Toutefois, certaines fonctionnalités spécifiées dans la norme ne sont actuellement pas réalisables.
En plus des diagrammes BPMN schématisés directement avec le logiciel, il est possible d'utiliser des fichiers XML d'extension *.bpmn*. Par ailleurs, Bonita permet d'importer des définitions de processus écrites dans le format standard XPDL 1.0 (format *.xpdl*). Les processus jBPM 3.2 sont également pris en charge mais ce format ne constitue pas un standard adopté par un organisme de standardisation, contrairement à BPMN et XPDL.
Ainsi, Bonita utilise une notation conforme à la norme BPMN pour son format propriétaire (.proc) et permet d'importer des processus définis avec les formats XML standards BPMN 2.0 et XPDL 1.0. Le produit obtient la note de 2 sur 2.

Enhydra Shark

Enhydra Shark a été conçu dès le départ avec un souci de conformité aux standards établis par la *Workflow Management Coalition* (WfMC). En effet, les processus sont définis selon le format WfMC-XPDL sans extensions propriétaires. Tout processus XPDL 1.0 défini avec n'importe quel outil tiers peut être ouvert dans l'éditeur *TWE* et exécuté par le moteur ES. Les règles d'exécution du moteur de flux sont conformes au standard. Nous pouvons toutefois regretter que le produit soit conforme à la version 1.0 de XPDL alors que la version 2.1 est sortie depuis 2008.

ES prend en charge le protocole *ASAP* (Asynchronous Service Access Protocol) du consortium OASIS qui normalise le démarrage et le suivi à distance de programmes. Plus particulièrement, l'extension Wf-XML, concernant les moteur de flux de travail est prise en charge. Ceci permet la communication de moteurs Enhydra avec d'autres moteurs de flux via le protocole ASAP (Interface 4 du modèle de référence WfMC). Les communications inter-moteurs englobent l'installation de définitions de processus et le démarrage d'instances de processus à distance.

Par ailleurs, la dénomination adoptée est conforme à celle établie par la WfMC (*Tool Agent*, Participant, liste d'items de travail, etc). Au delà de la simple dénomination, les implémentations, entre autres, des agents (tool agent) et de la journalisation sont conformes aux spécifications définies par la coalition (respectivement les interfaces 3 et 5 du modèle de référence WfMC). Des trois produits évalués, Enhydra Shark est probablement le moteur de flux le plus rigoureux dans sa conformité aux standards établis. Il obtient la note de 2 sur 2.

6.11 Performance et coût

6.11.1 Performance des moteurs de flux

jBPM

Les développeurs de jBPM ont implémenté un banc d'essai simple [19] qui déploie des processus jBPM et lance un programme en passant en paramètre un nombre d'itérations. À chaque itération, un processus, parmi ceux déployés précédemment, est aléatoirement choisi, démarré et suivi. Lorsque toutes les itérations s'achèvent, le temps total, les temps moyens par processus et le nombre de processus par heure sont calculés. Les processus choisis pour le banc d'essai mettent l'accent chacun sur une spécificité des moteurs de flux. Par exemple, un processus lance plusieurs tâches séquentielles qui ne font qu'envoyer un signal pour poursuivre l'exécution de l'instance. Un second processus teste une divergence et une convergence simple. Un troisième processus teste l'utilisation de variables de processus pour diriger le flux, etc.

La première mesure a été faite avec les configurations par défaut de jBPM et de Hibernate. La journalisation était activée et détaillée (Hibernate affichaient les requêtes SQL) et la mise en cache était désactivée.

La deuxième mesure fût réalisée en redirigeant l'affichage de la console dans un fichier. L'exécution résultante était 4.5 fois plus rapide.

La désactivation de l'affichage des requêtes SQL de Hibernate a permis d'améliorer les temps d'exécution de 8%.

La désactivation complète de la journalisation dans la base de données améliore les résultats précédents par un facteur de trois.

L'activation de la mise en cache de Hibernate permet d'obtenir une exécution plus rapide d'environs 5%.

La dernière configuration du banc d'essai a permis d'exécuter une moyenne de 401

---

19. http ://www.jorambarrez.be/blog/2008/04/14/short-jbpm-performance-showdown/ (dernière consultation le 09-11-10)

393 processus par heure. Chaque processus prenant en moyenne 8,97 millisecondes pour s'exécuter. Même si les définitions de processus utilisées sont minimalistes et ne représentent pas des scénarios réels, ce chiffre est un bon indicateur des performances de jBPM.

On note que le système est fortement dépendant du cadre Hibernate, qui gère l'ensemble des transactions de stockage des données en temps réel. Hibernate dispose de plusieurs mécanismes d'optimisation des transactions avec la base de données et est réputé pour être une bibliothèque de correspondance objet-relationnel performante. Un paramétrage avancé de l'outil nécessite toutefois une certaine expertise.

Concernant l'exécution de code Java ou de scripts, les performances de la machine virtuelle Java s'améliorent d'une édition à l'autre (amélioration de la gestion de la mémoire, du ramasse-miettes...).

Malheureusement, il n'existe pas à notre connaissance de bancs d'essais comparant les performances de jBPM avec d'autres systèmes de flux.

jBPM se base sur la technologie Java et le cadre Hibernate qui permettent d'obtenir de bonnes performances. Le système obtient une note de 1 sur 2 sur ce critère.

Bonita Open Solution

Les développeurs de Bonita utilisent des suites de tests en interne pour évaluer le fonctionnement du produit. Parmi ces tests, certains mesurent les performances du moteur de flux. Une interface *PerfTestCase* définit les méthodes *getBusinessArchive*, *deploy*, *launch*, *undeploy*, *getId* et *setId* servant à manipuler des instances de processus déployées à des fins de test.

La classe *StressPerfTest* lance un ensemble de processus (instances de *PerfTestCase*) et effectue un ensemble de mesures sur les temps d'exécution moyens, les meilleurs et les pires temps ainsi que le nombre d'exécutions réussies pour chaque instance déployée. Les mesures sont précises et distinguent les temps de mise en place (startup / warmup) des temps d'exécution. À la fin de l'exécution, les statistiques de test récoltées sont affichées.

En se basant sur ces informations, Bonita assure que les performances des dernières versions sont supérieures à celles des versions précédentes. Nous avons demandé aux développeurs de publier les résultats de leurs évaluations pour des volumes importants de processus déployés, mais n'avons pas obtenu de retour d'information.

Au delà des performances intrinsèques du moteur de flux, les temps d'exécution du système dépendent de plusieurs autres critères. Tout d'abord, le système s'exécute sur la machine virtuelle Java et dépend de ses performances. Par ailleurs, Bonita réalise plusieurs transactions avec les bases de données pour stocker et récupérer les informations sur les processus en cours, l'historique d'exécution, etc. La persistance des données se fait via le cadre Hibernate. En ajustant les paramètres de configuration relatifs à la persistance (mécanismes de récupération des données, utilisation du cache...), on obtient des variations importantes dans les temps d'exécution. De plus, la configuration de la journalisation influence les performances. En adaptant le niveau d'informations à journaliser pour chaque aspect du système (erreurs, avertissements, requêtes SQL...) on peut sensiblement raccourcir les temps d'exécution. Enfin, si les processus sont déployés dans un conteneur web ou sur un serveur d'applications, leurs performances influencent les temps d'exécution. Il est alors souhaitable d'ajuster leurs configurations respectives pour répondre aux besoins de l'utilisateur.

Malheureusement, nous ne disposons pas de résultats de bancs d'essais pour avoir une mesure précise du comportement de Bonita, en comparaison avec d'autres moteurs de flux. Étant donné que les temps d'exécution dépendent fortement de bibliothèques et de systèmes tiers et que ces composants sont hautement configurables, nous estimons qu'il est possible d'obtenir des performances honorables. La note accordée sur ce critère est de 1 sur 2.

Enhydra Shark

ES est un système hautement configurable. Les attributs du noyau même du moteur de flux peuvent être modifiés. Cette capacité de paramétrage permet d'adapter le produit

aux besoins et à la logistique de l'organisme client afin d'obtenir les meilleures performances.

De plus, ES peut mettre en cache des processus et des ressources. L'activation des paramètres *Cache.InitProcessCacheString=** et *Cache.InitiResourceCacheString=** dans les fichiers de configuration fait en sorte que les processus et les ressources sont créés à partir de données de la base de données, puis ils sont placés dans le cache. Il est possible de spécifier les identifiants des ressources et des processus à mettre en cache au démarrage du moteur de flux. Ainsi, les entités fréquemment utilisées peuvent être accessibles dès le démarrage, directement à partir de la mémoire vive. Ceci permet de réduire sensiblement les temps d'exécution.

D'autre part, les tailles de cache réservées pour les identifiants des activités et des processus sont configurables. Lorsqu'un processus ou une activité est créée, ES récupère de la couche de données son identifiant unique. En mettant les identifiants dans le cache, on réduit les communications du système avec sa base de données. Ci-dessous, un extrait du fichier de configuration relatif aux tailles de cache pour les identifiants :

```
# taille de cache generique
DODS.defaults.IdGenerator.CacheSize=100
# taille de cache pour les identifiants d'instances de processus
#DODS.IdGenerator._process_.CacheSize=100
# taille de cache pour les identifiants d'instances d'activit\'es
#DODS.IdGenerator._activity_.CacheSize=100
```

Par ailleurs, le noyau du moteur de flux peut être configuré pour définir une limite du nombre d'allocations au dessous de laquelle il n'est pas nécessaire de récupérer l'ensemble des ressources de la base de données. Le fait de ne charger qu'un nombre limité de ressources dans le système permet d'améliorer les performances du moteur. La configuration suivante fixe une limite de cinq allocations.

```
SharkKernel.LimitForRetrievingAllResourcesWhenCreatingAssignments=5
```

ES propose deux modèles de données pour stocker les variables de processus :

- Tous les types de données existent dans une seule table, y compris les types binaires

BLOB (Binary Large Object) utilisés pour persister des objets Java et des chaînes de caractères volumineuses.

- Une table contient tous les types de données sauf le type BLOB. Une seconde table, référencée par la première, est dédiée aux données de ce type.

Le second modèle peut améliorer les performances dans le cas où le système n'a pas à stocker un volume important d'objets Java ou de larges chaînes de caractères. Cette approche a été testée avec un SGBD Oracle et a donné de bons résultats selon la documentation d'ES.

En outre, l'utilisation des transactions permet d'accélérer les temps d'exécution. À titre illustratif, un utilisateur rapporte sur la liste de courriels publique d'ES que la récupération d'une liste de travail composée de 25 éléments a mis 4800 millisecondes sans l'utilisation des transactions et 1800 millisecondes avec les transactions, ce qui fait un gain de plus de 300%.

D'autres paramètres interviennent dans l'optimisation des performances, notamment les paramètres de journalisation, d'audit et de persistance.

Enfin, le contexte d'exécution influence les temps d'exécution. ES dépend de la machine virtuelle Java. S'il est déployé dans un conteneur web ou sur un serveur d'applications, un paramétrage adéquat de ces intergiciels permet d'améliorer les performances générales.

En définitive, contrairement aux moteurs de règles dont les stratégies d'exécution sont le facteur ultime de mesure de la performance, plusieurs paramètres influencent les temps d'exécution des moteurs de flux. Étant donné qu'on ne dispose pas de bancs d'essais comparant les produits évalués, nous attribuons à ES la même note que les autres produits, c'est à dire 1 sur 2.

6.11.2 Coûts en ressources matérielles

jBPM

Le système de gestion de flux jBPM fait intervenir plusieurs composants, chacun ayant son coût en ressources matérielles :

- Les bibliothèques Java formant l'API de jBPM ont une taille d'environs 1 MB. Toutefois, jBPM utilise plusieurs bibliothèques externes (Hibernate, pilotes du SGBD, archives des applications web...), dont certaines sont optionnelles. L'ensemble des bibliothèques nécessite 83.5 MB d'espace de stockage.
- En plus du moteur de flux, un système de gestion de bases de données doit être installé. Par défaut, jBPM utilise HyperSQL DB, qui est un SGBD léger, purement Java. Cette configuration peut s'avérer insuffisante dans le cadre d'un déploiement en entreprise. Un SGBD tel que MySQL ou PostgreSQL est alors nécessaire. Les besoins en espace de stockage et en mémoire vive de ces SGBDs dépendent du nombre d'utilisateurs et du volume des données.
- Par ailleurs, les applications web tel que la console jBPM et l'éditeur graphique Signavio doivent être déployées dans un conteneur web tel qu'Apache Tomcat. Si l'organisme client étend les applications jBPM ou en crée de nouvelles qui utilisent des composants Java d'entreprise (système de messagerie, EJB...), un serveur d'applications est alors nécessaire. jBPM fournit des scripts d'installation dans le serveur JBoss Application Server. La documentation du serveur indique qu'un processeur cadencé à 1GHz, une mémoire vive de 1.5 GB et un espace de stockage de 1.5 GB permettent de prendre en charge une application d'entreprise moyenne dans un cadre de production.
- Enfin, le plugiciel *eclipse* fournit avec jBPM nécessite l'installation de la dernière version de l'IDE (*eclipse* 3.5). Une machine avec un processeur Pentium récent et un minimum de 256 MB de RAM sont requis pour exécuter l'IDE.

Par conséquent, les coûts en ressources matérielles de jBPM varie en fonction du SGBD utilisé, de l'utilisation d'un conteneur web ou d'un serveur d'applications pour déployer les applications web et de l'utilisation ou pas de l'IDE *eclipse* (recommandé vu que

l'utilisateur aura probablement à manipuler du code java). Le noyau du système ne présente toutefois pas de coûts excessifs en ressources matérielles.
jBPM obtient une note de 1 sur 2 sur ce critère.

Bonita Open Solution

La documentation officielle de Bonita recommande l'utilisation d'un processeur cadencé à 1GHz et un minimum de 512 MB de mémoire vive. Les utilisateurs Windows sont encouragés à disposer d'au moins 1 GB de mémoire RAM pour éviter les ajustements de fichiers d'échange swap et obtenir de meilleures performances.
Concernant l'espace de stockage nécessaire, les bibliothèques Java *bonita-server* occupent 1.1 MB. Les bibliothèques *bonita-client*, utilisées pour se connecter à un serveur Bonita occupent quant à elles 514 KB. Toutefois, plusieurs dépendances à des bibliothèques externes sont requises, dont Hibernate, commons-logging (journalisation), Groovy et H2 (système de gestion de bases de données par défaut).
L'éditeur Bonita, basé sur la plateforme *eclipse*, occupe 336.4 MB lorsqu'il est installé. Si son installation n'est pas requise, elle est fortement recommandée. L'éditeur fournit une plate-forme pour définir graphiquement les processus et concevoir les formulaires web, entre autres.
Le système de gestion de bases de données par défaut H2 est léger. Toutefois, il est possible qu'un organisme ait besoin d'un SGBD plus robuste. Les besoins en espace de stockage et en mémoire vive de ces SGBDs dépendent du nombre d'utilisateurs et du volume de données.
Enfin, le déploiement dans un conteneur web ou sur un serveur d'application rajoute des coûts matériels. Comme indiqué précédemment, un serveur d'applications tel que JBoss Application Server recommande 1.5 GB de mémoire vive et un processeur cadencé à 1 GHz. Les besoins matériels pour un conteneur web tel qu'Apache Tomcat sont moindres. Les exigences matérielles de Bonita sont donc variables. Globalement, elles restent raisonnables et dans la moyenne des exigences d'un système informatique d'entreprise. Le produit obtient une note de 1 sur 2.

Enhydra Shark

L'installation complète du système de gestion de flux ES incluant le moteur de flux, l'éditeur *TWE* et les applications d'administration occupe 574.7 MB d'espace disque. L'éditeur *TWE* a une taille de 31.5 MB. Dans le cadre d'une utilisation dans une application Java, les bibliothèques client ne font que 121.3 KB. Le total des bibliothèques, incluant les outils tiers occupe 42.1 MB.
La documentation officielle du produit ne spécifie pas d'exigences matérielles. Les composants du système étant configurables, il est possible de choisir des composants légers. Le système peut par exemple être déployé sur un conteneur web Apache Tomcat ou Jetty et utiliser HyperSQL DB ou PostgreSQL comme SGBD. Des services optionnels tel que la journalisation et l'audit peuvent être désactivés. Ainsi, le système peut être mis à l'échelle pour s'adapter aux contraintes matérielles.
Les exigences matérielles sont donc variables. La pile de produits typiques à installer présente des besoins raisonnables en ressources. Une note de 1 sur 2 est accordée sur ce critère.

6.12 Prise en main

6.12.1 Documentation

jBPM

jBPM présente une documentation :

- destinée aux utilisateurs : Le guide utilisateur[20] décrit le langage jPDL et illustre les scénarios d'utilisation de base du système. L'installation du moteur de flux, le déploiement des applications web, l'installation du plugiciel *eclipse* et la configuration de la base de données sont expliqués. Le document décrit le déploiement et l'instanciation de processus ainsi que la création et le déclenchement de tâches. Une

20. http ://docs.jboss.com/jbpm/v4/userguide/html_single/ (dernière consultation le 09-11-10)

section consacrée au langage jPDL revient sur les différents éléments de flux (processus, tâche, événement, sous-processus, tâche HQL...) et illustre leur utilisation par des exemples clairs et simples. Le guide utilisateur est pédagogique et les exemples sont illustratifs. Une connaissance basique de Java et de XML sont toutefois requis.

- destinée aux développeurs : Le guide pour développeurs[21] explique des options de paramétrage avancées qui sont encore au stade expérimental. Le contenu s'adresse aux développeurs expérimentés qui visent une utilisation avancée du système.

  Un chapitre "incubation" explique les fonctionnalités qui devraient être officiellement prises en charge dans les prochaines éditions de jBPM. Un chapitre "BPMN" illustre l'implémentation réalisée par jBPM de la notation BPMN. Plusieurs chapitres expliquent le fonctionnement interne de la machine virtuelle de processus PVM. Des illustrations permettent de visualiser les flux d'exécutions des processus dans la machine virtuelle. Le reste du guide revient sur des cas d'utilisation avancés ainsi que l'éditeur graphique Signavio.

En plus des guides officiels, jBPM dispose d'un wiki collaboratif[22] où des articles soumis par les développeurs du projet ou par la communauté sont organisés par thèmes. Les utilisateurs peuvent attribuer une note à chaque article, comme ils peuvent le télécharger sous format PDF.

D'autre part, (Salatino, 2010) est un livre destiné aux développeurs désirant utiliser jBPM dans une architecture Java entreprise. Le livre contient plusieurs exemples d'implémentations conçues pour un environnement de production. L'implémentation interne du système est expliquée en détail. Le livre suppose une bonne connaissance du langage Java et des bibliothèques externes tel que Hibernate et log4j. Toutefois, le document se base sur l'ancienne version 3.2 de jBPM.

jBPM étant principalement destiné aux développeurs Java, l'ensemble de la documentation contient plus ou moins de détails techniques. Ceci dit, le contenu est clair, de bonne qualité et illustré par plusieurs schémas et exemples de code. Par conséquent,

---

21. http ://docs.jboss.com/jbpm/v4/devguide/html_single/ (dernière consultation le 09-11-10)

22. http ://community.jboss.org/en/wiki (dernière consultation le 09-11-10)

nous attribuons la note de 2 sur 2 sur ce critère.

Bonita Open Solution

Bonita fournit un document introductif *Quick Start Guide* qui se base sur un exemple de processus de vente en ligne pour illustrer les principales fonctions du produit. Le document est présenté sous forme de tutoriel, guidant progressivement l'utilisateur dans l'implémentation de l'exemple. Chaque étape est expliquée avec des captures d'écran et une explication détaillée des opérations à réaliser. Le processus exemple comporte des portes de convergence, des transitions conditionnées, des étapes automatiques et des étapes humaines. Ces dernières fournissent un contexte pour expliquer la conception de formulaires et l'assignation de tâches aux ressources.

Par ailleurs, un guide utilisateur *User and Reference Guide* explique en détail l'utilisation du produit. Le document revient en détail sur la conception des processus et la manipulation de l'ensemble de leurs composants (étapes, événements, connecteurs...). Un chapitre est consacré à la création et à la personnalisation des formulaires. Un autre chapitre illustre l'utilisation de l'application web User XP. Les procédures d'installation du système, de configuration des bases de données et de gestion de l'authentification sont brièvement décrites.

Un troisième document, *Connector Reference Guide*, illustre la configuration et l'utilisation de tous les connecteurs prédéfinis Bonita.

L'ensemble de la documentation est gratuitement accessible. Elle est également à jour avec la dernière version 5.2 de Bonita. Une approche par l'expérimentation et de nombreuses illustrations en font un support de qualité pour les utilisateurs métier. Les détails techniques n'y sont pas évoqués. Les développeurs désirant maîtriser les mécanismes sous-jacents du système peuvent se référer à la documentation de l'API Bonita.

En définitive, le produit fournit une documentation accessible aux utilisateurs métier. On peut justifier l'absence de détails techniques par le fait que la plate-forme Bonita abstrait la définition des processus, des activités et des interfaces graphiques, couvrant ainsi tous les aspects d'un système de gestion de flux de travail. Par ailleurs, les développeurs

désirant maîtriser les détails techniques peuvent consulter la documentation de l'API Bonita. Une note de 2 sur 2 est accordée sur ce critère.

Enhydra Shark

Enhydra Shark fournit un document introductif *Quick Start Example* où un processus fournit avec le produit est instancié à partir de l'application d'administration. À travers cet exemple, les concepts de participants, de variables de processus et de liste d'items de travail sont notamment expliqués. Le document intitulé *Enhydra Shark* revient sur les procédures d'installation et de configuration du produit, expliquant entre autres la manière de modifier le SGBD utilisé pour la persistance des données. *Enhydra Shark Admin*, comme son nom l'indique, documente les applications d'administration fournies.

L'éditeur *TWE* possède lui aussi son document de référence. Il s'agit d'un texte de 113 pages qui explique exhaustivement les fonctionnalités de l'éditeur en utilisant de nombreuses captures d'écran. Toutes les opérations sont détaillées pour les novices, y compris l'ouverture et la sauvegarde de fichiers XPDL.

Plusieurs autres documents d'une dizaine de pages chacun expliquent des modules spécifiques du produit (communication avec les dépôts LDAP, *Tool Agents*, attributs étendus dans les définitions de processus XPDL, déploiement du moteur de flux comme EJB...)

Le guide de démarrage, la documentation de l'éditeur et celle des applications d'administration sont destinés aux experts métier. La documentation qui concerne la configuration du système, la définition d'agents ou encore le déploiement dans des serveurs d'applications est destinée aux développeurs.

Globalement, la documentation est écrite dans un style concis et efficace. Plusieurs bouts de codes viennent illustrer les fonctionnalités du produit. L'information est structurée par thèmes et on localise rapidement l'information recherchée. Une note de 2 sur 2 est attribuée sur ce critère.

6.12.2 Convivialité

jBPM

La console jBPM, l'éditeur graphique de processus Signavio et le plugiciel *eclipse* fournissent des interfaces ergonomiques, dont le fonctionnement est rapidement maîtrisé par l'utilisateur. Étant donné que l'installation des différents composants demande une certaine compétence technique (déploiement d'une application web, initialisation du schéma de base de données relationnelle, installation d'un serveur d'applications...), jBPM fournit des scripts *ant* se chargeant d'automatiser l'ensemble des procédures. Plusieurs cibles *ant* existent, allant du téléchargement de l'IDE *eclipse*, jusqu'au démarrage du SGBD, en passant par le déploiement d'applications web. Les différentes procédures sont expliquées dans la documentation officielle du produit. Un utilisateur métier devrait donc parvenir à installer l'ensemble des composants du système, en suivant les procédures indiquées.

Ainsi, malgré son fort couplage à la technologie Java, jBPM parvient à fournir des moyens d'installation accessibles aux utilisateurs métiers et des interfaces ergonomiques pour une utilisation basique du système.

Toutefois, les scénarios d'utilisation avancés requièrent un certain niveau d'expertise en programmation. La réalisation de la majorité des contrôles de flux, de l'allocation de ressources et de la manipulation de données nécessitent l'implémentation de code Java. De plus, le paramétrage des services externes tel que la base de données ou le service de journalisation impliquent la manipulation de fichiers de configuration Hibernate et log4j, entre autres.

Ainsi, la convivialité de l'installation et d'une utilisation basique est contre-balancée par les efforts de programmation nécessaires dans le cadre d'une utilisation avancée. La note attribuée sur ce critère est de 1 sur 2.

Bonita Open Solution

L'installation du système Bonita se fait en désarchivant un fichier compressé. La solution fournit un éditeur visuel puissant et intuitif pour définir les processus. Elle utilise un conteneur web installé par défaut avec l'éditeur et gère automatiquement le déploiement des applications web générées à partir des définitions de processus. La définition des connecteurs et des propriétés avancées du processus est facilitée par des assistants qui abstraient les détails d'implémentation et accélèrent l'édition. De plus, un module de création de formulaires permet de réaliser rapidement les interfaces de communication avec les participants au flux de travail. Bonita incorpore également des éditeurs Java et Groovy munis de différents outils d'aide au développement (coloration syntaxique, auto-complétion...). Enfin, l'application User XP constitue une couche présentation riche et ergonomique reprenant le modèle familier des boîtes de courriel. Ainsi, en plus d'être facile à installer, Bonita fournit des interfaces conviviales et faciles d'accès pour l'ensemble des fonctionnalités du système. Le produit obtient une note de 2 sur 2.

Enhydra Shark

L'installation d'Enhydra Shark consiste à désarchiver un fichier contenant l'ensemble des composants du produit (éditeur, bibliothèques, applications d'administration, etc). La configuration par défaut du système permet de l'utiliser immédiatement. L'application d'administration accède par défaut au moteur comme une bibliothèque POJO. Les implémentations des patrons de flux de travail sont automatiquement importées dans l'éditeur *TWE* et plusieurs exemples permettent de saisir rapidement le fonctionnement du système. Les exemples sont variés et exposent différents aspects du produit (agents JavaScript, agents Java, sous-flux, etc).

L'éditeur en lui-même est ergonomique. L'ensemble des assistants et la représentation graphique des processus facilitent l'utilisation du système. Plusieurs configurations sont fournies pour utiliser le moteur de flux avec le protocole Corba, RMI ou comme service web. De même, les schémas de base de données pour plusieurs SGBDs (neufs au total)

accompagnent l'installation d'ES.

Les applications d'administration sont également intuitives et faciles d'accès. Un message expliquant le rôle de chaque bouton de l'interface graphique est déclenché lorsque le curseur est placé dessus (tooltip). L'organisation des fonctions par catégories permet à l'utilisateur de se repérer rapidement.

Ainsi, ES est facile à installer. Son éditeur et ses applications d'administration sont ergonomiques et bien documentées. Une note de 2 sur 2 est accordée sur ce critère.

6.12.3 Courbe d'apprentissage

jBPM

En dehors de la modélisation des processus, la programmation Java est un prérequis indéniable pour pouvoir utiliser jBPM. De plus, les développeurs Java doivent maîtriser l'API de jBPM afin d'utiliser les services d'instanciation de processus, de gestion de tâches et de définition des ressources, entre autres.

La modification du comportement par défaut du système nécessite dans plusieurs cas un important effort de codage.

La réalisation de scénarios impliquant l'exécution concurrente ou l'instanciation d'un volume important de processus requièrent une compréhension approfondie du fonctionnement interne de la machine virtuelle de processus.

Par ailleurs, le paramétrage des bases de données et des transactions de persistance, ainsi que la journalisation des événements nécessitent la manipulation de Hibernate et de log4j.

En définitive, jBPM est un système flexible. L'architecture de base fournie par le système ne constitue qu'un point de départ, à partir duquel les utilisateurs doivent programmer leurs processus.

Ainsi, le système présente une courbe d'apprentissage assez forte et obtient une note de 1 sur 2 sur ce critère.

Bonita Open Solution

La modélisation des processus avec Bonita ne requiert pas l'apprentissage d'un langage informatique, puisqu'elle se fait avec un éditeur visuel. De même, les formulaires servant à échanger des données avec les participants aux flux de travail peuvent être définis dans un autre éditeur graphique. L'utilisateur n'a ainsi pas à manipuler de langage de programmation web. Les blocs de construction des processus et des formulaires web sont directement accessibles via des palettes de composants.
Par ailleurs, plusieurs connecteurs prédéfinis permettent de réaliser des activités complexes tel que le partage de documents, l'exécution de règles Drools ou encore l'accès à une base de données, simplement en fournissant les informations demandées par l'assistant du connecteur. Enfin, la réalisation de comportements complexes tel que la multi-instanciation et la répétition de tâches se fait en cochant des boutons radio et en fournissant les données demandées. Enfin, le déploiement des applications web est géré automatiquement par Bonita.
Tout en octroyant aux développeurs la possibilité d'utiliser du code Groovy et Java et aux programmeurs web celle de modifier le code HTML et CSS des formulaires, Bonita permet aux utilisateurs métier de rapidement concevoir et utiliser un système complet de gestion de flux sans avoir à apprendre de technologies particulières. Par conséquent, la courbe d'apprentissage de Bonita est faible. Le produit obtient une note de 2 sur 2.

Enhydra Shark

Grâce à la représentation graphique, la modélisation des processus ne requiert pas de connaître XPDL. Les concepts définis par la WfMC doivent toutefois être assimilés (agent, participant, etc), ce qui est normal pour un système de gestion de flux de travail. Les applications d'administration facilitent sensiblement la gestion des processus, des utilisateurs et des items de la liste de travail. Ainsi, en interagissant avec le système à travers les applications qu'il fournit, un utilisateur est capable de définir et de gérer des processus sans manipuler l'API Java ou le code XPDL. La modification du SGBD

utilisé est également faite sans effort, grâce aux fichiers de configuration prédéfinis.
D'autres procédures plus avancées nécessitent un effort d'apprentissage. Nous en citons :
- L'inclusion du moteur de flux dans une application personnelle.
- La modification des paramètres par défaut afin d'optimiser les performances.
- Le changement de composants tel que le système de journalisation ou d'audit.
- Le déploiement comme application EJB ou service web.
- L'accès au système avec les protocoles RMI ou CORBA.
- ...

Étant donné qu'ES utilise des technologies répandues dans la programmation Java d'entreprise et que les paramètres de configuration sont exhaustivement expliqués, un développeur moyen devrait être capable de rapidement maîtriser le produit.
Ainsi, la courbe d'apprentissage est faible pour les experts métiers et les développeurs, chacun pour les opérations qui lui sont destinées. Une note de 2 sur 2 est accordée sur ce critère.

6.13 Évaluation sur l'aspect logiciel libre

6.13.1 Communauté

jBPM

jBPM dispose d'un forum utilisateurs avec 3670 participants enregistrés et un total de 8000 messages, ainsi qu'un forum développeurs avec 431 participants inscrits et un total de 1450 messages. Une trentaine de messages par semaine sont créés dans le forum utilisateur contre une moyenne de 6 messages par semaine dans le forum des développeurs. De plus, une plate-forme de suivi des bogues[23] (issue tracker) permet aux utilisateurs de rapporter des bogues et de voter pour augmenter la priorité de résolution d'un bogue ou d'implémentation d'une fonctionnalité.
Une liste de courriel (mailing list) nommée jbpm-cvs permet d'informer les utilisateurs

23. https ://jira.jboss.org/browse/JBPM (dernière consultation le 09-11-10)

des "commits" de code effectués par les développeurs.

Par ailleurs, des groupes d'utilisateurs de JBoss en général existent partout dans le monde[24]. Les membres de la communauté s'y réunissent et traitent des projets JBoss, dont jBPM.

Sur le site de téléchargement du produit[25], jBPM 4 (toutes versions confondues) enregistre 81 434 téléchargements. jBPM3, enregistre 562 827 téléchargements depuis juin 2005.

Enfin, un écosystème s'est formé autour du projet. Par exemple, l'application web Teamwork[26] est basée sur jBPM. Il s'agit d'un produit de gestion de projets, utilisé notamment par la société Canonical (créateur du système d'exploitation Ubuntu) et dans le milieu universitaire. Le produit a obtenu le prix Jolt Productivity Award en 2007.

Ainsi, jBPM possède une importante communauté, des forums utilisateurs et développeurs, une plate-forme de suivi de bogues publique et des groupes d'utilisateurs. Le projet affiche un nombre élevé de téléchargements et a été utilisé pour créer d'autres produits. Il mérite donc une note de 2 sur 2 sur ce critère.

Bonita Open Solution

Bonita revendique plus de 60 000 visiteurs uniques par mois sur le site web du produit, provenant de plus de 200 pays. Bonita dispose d'un forum public comprenant 1 230 utilisateurs inscrits. Ces derniers sont très actifs, avec une vingtaine de messages échangés par jour. Le forum est structuré en six sections :

- Installation : Questions sur l'installation et le déploiement de Bonita.
- Utilisation : Questions sur l'utilisation des fonctionnalités du produit.
- Développement de contributions : Aide pour les membres de la communauté désirant développer des contributions.

---

24. http ://www.jboss.org/usergroups (dernière consultation le 09-11-10)

25. http ://sourceforge.net/projects/jbpm/files/

26. http ://www.twproject.com/ (dernière consultation le 09-11-10)

- Commentaires sur les contributions : Questions des utilisateurs des contributions.
- Suggestions aux développeurs de Bonita : Les membres de la communauté font part des fonctionnalités qu'ils désireraient voir implémentées dans Bonita.
- Questions non techniques : Discussions générales sur le produit et les systèmes de gestion de processus.

Concernant les contributions, 46 ont déjà été fournies alors que la soumission de contributions n'est apparue qu'au début de l'année 2010. La communauté a effectué à ce jour plus de mille téléchargements de contributions. Quant au système Bonita Open Solution, il a été téléchargé plus de 300 000 fois selon les développeurs.

Par ailleurs, une plate-forme de suivi de bogues publique permet aux membres de la communauté de rapporter des dysfonctionnements, de les commenter et de suivre l'évolution ou encore de prendre en charge la résolution des bogues.

Les développeurs organisent des conférences web avec la communauté anglophone et francophone du produit. Les utilisateurs y ont l'occasion de poser en direct leurs questions. Des tutoriels sur l'utilisation du produit sont également diffusés pendant les conférences.

De plus, Bonita dispose d'un blog pour informer la communauté sur l'évolution du produit. Une chaîne youtube[27] publie des tutoriels et les enregistrements des conférences web passées. Les vidéos ont été visualisées plus de 30 000 fois jusqu'à présent.

Ainsi, la communauté a une importance primordiale. De nombreux canaux de communication permettent d'instaurer un échange constant et procurent une réelle proximité. Les utilisateurs sont appelés a participer à tous les niveaux et leur avis influencent l'évolution du produit. Une note de 2 sur 2 est accordée sur ce critère.

27. http ://www.youtube.com/user/BonitaSoft (dernière consultation le 09-11-10)

Enhydra Shark

Enhydra Shark ne dispose pas d'une communauté importante. Le forum du produit [28] présente une activité faible. Sur les 8 derniers mois, une moyenne de 4 messages par mois ont été échangés.

Le produit ne dispose pas d'une plate-forme de suivi de bogues publique. Le seul moyen de remonter des informations sur d'éventuelles défaillances est d'envoyer des messages sur le forum.

ES revendique certaines contributions de la communauté. Par exemple, elles accordent à un utilisateur l'implémentation de l'API *LimitAgent*, qui est une implémentation d'un chronomètre simple. Un autre membre de la communauté a fourni les fonctionnalités d'envoi de courriels et d'échange de messages JMS (système de messagerie Java) à l'agent Java *JavaClassToolAgent*. Un troisième utilisateur est accrédité pour l'implémentation de l'allocation aux ressources selon l'historique. Les noms des contributeurs et leurs adresses de courriel sont cités. Plus de vingt contributions sont affichées. Toutefois, plusieurs d'entre elles consistent en simples suggestions ou signalisations de bogues.

Les développeurs du produit ne tiennent pas de blog. Il n'existe pas de plate-forme de documentation collective (wiki). En définitive, très peu d'échanges sont faits avec la communauté. Une note de 1 sur 2 est accordée sur ce critère en raison de l'existence d'un forum et des contributions de la communauté.

6.13.2 Publications

jBPM

jBPM 1.0 fût publié en décembre 2003. Une mise à jour mineure, jBPM 1.0.1, est sortie quatre mois plus tard.

jBPM 2.0 a eu quatre versions, la première étant sortie en octobre 2004.

Le plugiciel *eclipse*, nommé jBPM GPD, a vu le jour en juin 2005 et est actuellement

28. http ://old.nabble.com/Enhydra-f2472.html (dernière consultation le 09-11-10)

publié avec le système de flux. En tout, 26 versions du plugiciel ont été mises à disposition du public.
La version 3.0 et la version 3.2.6 sont respectivement la première et la dernière version de jBPM 3. La première est parue en juin 2005 et la version 3.2.6 en mars 2009. Entre les deux, 20 mises à jours mineures ont été publiées.
jBPM BPEL 1.1 est sorti en 2007, suivi d'une mise à jour en 2008. Enfin, jBPM 4 est la dernière mouture du logiciel. La version initiale 4.0 est sortie en juillet 2009 et la version actuelle 4.3, qui constitue la troisième mise à jour mineure, est parue en décembre 2009. On remarque que la fréquence de publication du produit est élevée. Entre chaque mise à jour majeure, plusieurs mises à jour mineures sont publiées.
Une note de 2 sur 2 est accordée sur ce critère.

Bonita Open Solution

Se basant sur des recherches démarrées à l'institut français INRIA, la première version de Bonita fût réalisée en 2001. Après Bonita 1, sept mises à jour mineures ont suivi entre les années 2003 et 2005. Le produit proposait déjà des interfaces graphiques pour la définition et l'exécution des processus. En juin 2006, Bonita v2 a vu le jour, apportant entre autres la prise en charge du standard XPDL et une réécriture des mécanismes d'itération. Bonita v2.1 est apparu six mois plus tard, ajoutant le versionnage de processus et l'intégration aux serveurs d'applications JBoss AS et Jonas. En février 2007, une nouvelle version a apporté un éditeur graphique conforme à XPDL ainsi qu'une application web nommée *BPM Console* et un éditeur de formulaires web. La version 3.1, parue en octobre de la même année, a rajouté une API de gestion de l'historique des processus. La version 4.0, nommée Nova Bonita, fût publiée en octobre 2008. L'amélioration de l'éditeur graphique et des applications web figurent parmi ses nouveautés. Deux mises à jours mineures plus tard, sort Bonita 5.0. L'éditeur Bonita Studio, l'application User XP et les connecteurs sont autant de nouveautés qui font de cette mise à jour une avancée majeure du produit. Entre avril et juin 2010, 3 mises à jour mineures sont publiées, dont la version actuelle 5.2.

Présentement, le code en cours de développement est accessible à partir d'un dépôt SVN [29]. L'activité y est importante et plusieurs mises à jours sont faites, parfois dans un intervalle de 24 heures.

Ainsi, au cours de ses 9 ans d'existence, Bonita a publié 5 mises à jour majeures et une quinzaine de mises à jour mineures. Au fil des publications, le produit s'est enrichi de plusieurs fonctionnalités et a bénéficié de la réécriture de nombreux composants. Le développement est actuellement actif et de nouvelles mises à jour devraient apparaître prochainement.

Une note de 2 sur 2 est accordée sur la fréquence de publication.

Enhydra Shark

La première version Enhydra Shark 1.0 fût publiée en septembre 2004 après quelques versions beta. Le produit comprenait les APIs d'audit et d'administration (mise à jour de paquetages, allocation de tâches, etc). Les agents de gestion de courriel et de services web SOAP existaient déjà. Shark 1.1 a apporté son lot de nouvelles fonctionnalités et d'améliorations de performance. La version 1.1-2 a principalement fourni des corrections de bogues et quelques modifications au cadre de persistance DODS.

Shark 2.0-beta1 fût réalisé en mai 2006. Les principaux ajouts de cette version sont la prise en charge globale des transactions JTA, un découplage du moteur de flux des interfaces CORBA, EJB et services web ainsi qu'une meilleure gestion des variables, entre autres. Shark 2.0-beta2 est paru en juin 2006, suivi de 6 autres versions beta. La dernière version beta, Shark 2.0-beta8 a été réalisée en mai 2007. Ces publications ont notamment amélioré la gestion de la liste d'items de travail, la gestion des exceptions et l'API d'administration. Les applications d'administration Swing et Web ont été incluses dans le produit et plusieurs corrections de bogues ont été effectuées. La version 2 finale a amélioré l'agent XSLT et certaines bibliothèques ont été mises à jour. Les mises à jour mineures 2.0-2 et 2.1-1 ont suivi. En décembre 2007, Enhydra Shark 2.2-1 fût réalisé

29. http ://www.bonitasoft.org/websvn (dernière consultation le 09-11-10)

avec la possibilité de lier les listes de travail à l'application MS Outlook via des services web. Enhydra Shark 2.3-1 a été publié en novembre 2008, suivi des versions 2.4-1 et 2.5-1 en 2009. La même année, la version 3.0-1 d'Enhydra Shark, désormais appelé Together Workflow Server, est sortie. La licence libre est passée à la version 3 de GPL. Quelques mises à jour mineures ont suivi, à savoir 3.1-1 et 3.1-2, apparue en juin 2010.
Cet historique des publications montre que le produit a été actif tout au long de son existence. Même s'il ne compte que trois versions majeures, plusieurs mises à jour mineures ont apporté des améliorations conséquentes. Si nous incluons les publications mineures, Enhydra Shark a connu plus d'une vingtaine de publications au cours de ses 7 ans d'existence. Le rythme de publication et la nature des améliorations apportées à chaque version valent au produit une note de 2 sur 2.

6.13.3 Longévité

jBPM

La première publication de jBPM (version 1.0) fût publiée en décembre 2003. Depuis, quatre versions majeures et une trentaine de mises à jour mineures ont vu le jour. La version 3 du produit est la plus utilisée, étant donné qu'elle existe depuis 2005. La dernière version, 4.3, publiée fin 2009, est tout aussi populaire et enregistre déjà plus de 34 000 téléchargements.
Toutefois, le futur du projet est incertain. Les deux développeurs principaux ont quitté JBoss et travaillent sur leur propre système de gestion de flux, Activiti, sous la direction de la société Alfresco[30]. Même si un nouveau développeur principal a été nommé pour jBPM et qu'un plan de développement a été publié, de sérieux doutes subsistent quand à la pérennité du projet. Il existe un réel risque qu'une partie de la base d'utilisateurs de jBPM migre vers le nouveau projet Activiti, lui aussi libre et gratuit.
Ainsi, jBPM affiche une longévité de 7 ans et une popularité croissante. Toutefois, le départ des deux principaux développeurs du produit menace son avenir. Une note de 1

30. http ://www.activiti.org/ (dernière consultation le 09-11-10)

sur 2 est accordée sur ce critère.

Bonita Open Solution

La première publication de Bonita date de l'année 2001. La popularité du produit fût croissante. En 2008, Bonita 4 atteint plus de 100 000 téléchargements et compte 30 clients en entreprise. En juin 2009, la société BonitaSoft est créée et compte une communauté de plus de 300 membres. En janvier 2010, Bonita Open Solution, appelée également Bonita 5.0 est publiée. Cette nouvelle mouture est téléchargée plus de 180 000 fois. Six mois plus tard, le nombre de téléchargements dépasse les 300 000 et la communauté compte plus de 1 000 membres.
Bonita jouit donc de 9 ans d'évolution et affiche une popularité croissante. Le produit obtient la note de 2 sur 2.

Enhydra Shark

Le projet Enhydra Shark fût enregistré dans la forge du consortium OW2 (communauté de développement d'intergiciels) en juillet 2003[31]. Le produit compte donc 7 ans d'existence. Il est marqué comme *stable / en production* sur le site de la forge. En plus d'être mature, ES est activement développé. La version la plus récente est effectivement parue en juin 2010 et le site du projet est passé du domaine *enhydra.org* à *together.at*. Ces événements récents indiquent que le développement du produit est toujours actif. Nous regrettons toutefois que la norme suivie soit toujours XPDL 1.0, rendue obsolète par la version 2.1. Une note de 2 sur 2 est toutefois accordée sur ce critère.

6.13.4 Licence

jBPM

Le projet jBPM adopte des licences différentes selon le composant :

31. http ://forge.ow2.org/projects/shark (dernière consultation le 09-11-10)

- Licence LGPL (GNU Lesser General Public License) pour le moteur de flux : La principale différence entre les licences GPL et LGPL est que cette dernière peut être utilisée avec des programmes ayant une licence non GPL, qu'elle soit libre ou propriétaire. Le produit résultant peut être distribué sous n'importe quelle licence s'il ne s'agit pas d'une oeuvre dérivée du produit LGPL. Sinon, les utilisateurs doivent avoir le droit de modifier le produit pour leur usage personnel.
- Licence EPL (Eclipse Public License) pour le plugiciel *eclipse* : La licence EPL est une licence libre moins restrictive que la licence GPL. L'utilisateur d'un programme sous licence EPL a le droit de l'utiliser, le modifier, le copier et le distribuer, en étant dans certains cas obligé de publier ses propres changements. EPL est approuvée par l'Open Source Initiative et listée comme licence libre par la Free Software Foundation.
- Licence MIT pour le modélisateur web Signavio : La licence MIT est une licence libre qui a vu le jour au Massachusetts Institute of Technology (MIT). La licence permet de réutiliser le logiciel dans des produits propriétaires à condition que la licence soit distribuée avec le produit. La licence GPL permet de combiner et de redistribuer des logiciels sous licence MIT. Les deux licences sont dites compatibles.

Qu'il s'agisse de LGPL, EPL ou MIT, les licences utilisées par jBPM sont toutes libres et répondent aux exigences de la Chaire. Une note de 2 sur 2 est accordée.

Bonita Open Solution

Bonita Open Solution est publié avec une licence GNU General Public License v2 (GPLv2).

La licence GPL fût créée par Richard Stallman, l'initiateur du mouvement de logiciels libres, en 1989. La version 2 de la licence fût réalisée en juin 1991.

GPL est né du besoin d'unifier des licences libres similaires mais spécifiques chacune à un programme donné. Cette standardisation a levé les incompatibilités entre licences libres et a permis le partage de code entre différents projets.

La licence précise que ses termes et conditions doivent être disponibles pour chaque utilisateur recevant une copie d'un travail sous licence GPL. L'utilisateur acceptant les

termes est capable de modifier, de copier et de redistribuer le travail ainsi que toutes ses versions dérivées. Il lui est permis de vendre le produit ou de le distribuer gratuitement. Ceci distingue la licence GPL d'autres licences libres qui interdisent la redistribution commerciale. La fondation de logiciel libre (FSF) justifie ce choix par le fait que *free* dans *free software* signifie libre et non pas forcément gratuit.
GPL est une licence populaire. Elle est utilisée pour plus de la moitié des projets libres.[32] [33]. Le noyau Linux et le compilateur GCC figurent parmi les programmes GPL les plus connus.
La licence utilisée par Bonita provient de la fondation de logiciels libres et est naturellement conforme aux exigences de la Chaire. Le produit obtient donc une note de 2 sur 2 sur ce critère.

Enhydra Shark

Enhydra Shark propose deux types de licences : une licence libre GPLv3 pour les produits clients à code ouvert et une licence commerciale Together Software License V2 pour les produits à code fermé. La version 3 de la licence GPL, réalisée en Juin 2007, apporte principalement des changements par rapport aux patentes logicielles, à la définition du terme *code source*, aux restrictions matérielles par rapport aux modifications logiciels et à la compatibilité avec les autres licences libres. Les contributeurs aux logiciels sous licence GPLv3 peuvent désormais ajouter leurs propres termes et conditions.
Enhydra Shark précise que les projets qui utilisent ses composants sous licence GPLv3 doivent également être à code ouvert et utiliser GPLv3, comme stipulé par la licence.
Les composants à code ouvert utilisé par ES tel que l'outil de journalisation log4j gardent leurs licences originelles.
Le produit obtient une note de 2 sur 2 pour ce critère car la licence GPL est une licence

---

32. http ://www.dwheeler.com/frozen/sourceforge-stats-20031110.html (dernière consultation le 09-11-10)

33. http ://www.dwheeler.com/essays/gpl-compatible.html (dernière consultation le 09-11-10)

conforme aux exigences de la Chaire.

6.13.5 Aide communautaire

jBPM

Dans le forum des utilisateurs, les développeurs du projet interviennent régulièrement pour répondre aux questions. Il arrive souvent que plus d'un développeur réponde à une même question. Les délais de réponse sont généralement courts. Une barre d'expérience et un compteur du nombre de messages publiés par chaque membre du forum contribuent à encourager la participation, que ce soit pour poser des questions ou pour apporter son aide.

Plusieurs fonctionnalités ont été rajoutées dans jBPM pour répondre à des besoins exprimés par les utilisateurs du produit. Les développeurs incitent la communauté à lui exposer les cas d'utilisation en entreprise qu'elle rencontre afin d'ajuster le système en conséquence. À titre d'illustration, la fonction de migration de processus est à l'origine une revendication de la communauté jBPM.

L'aide communautaire est donc de qualité. Les développeurs du projet répondent aux questions des utilisateurs via les forums et essaient d'implémenter les fonctionnalités dont ils expriment le besoin via la plate-forme de suivi de bogues et les forums.

Une note de 2 sur 2 est attribuée à l'aide communautaire de jBPM.

Bonita Open Solution

Les forums Bonita comptabilisent plus de 10 000 messages échangés. Les développeurs interviennent fréquemment et s'assurent que l'utilisateur est satisfait de leurs réponses. L'aide est gratuitement fournie pour toutes les questions sur l'installation et l'utilisation du produit ainsi que sur les contributions de la communauté. Un membre du forum est capable de s'inscrire à un sujet pour recevoir des notifications par courriel au cas où une réponse est publiée. Les statuts des utilisateurs sont également affichés, à savoir qu'un membre inscrit actuellement connecté sur le forum est marqué *en ligne*.

Dans le cadre de notre évaluation, nous avons posé sept questions sur le forum, auxquelles nous avons obtenu des réponses dans un délai d'une semaine. La plupart des questions ont obtenu une réponse le jour même ou à deux jours d'intervalle.
D'autre part, le développement de Bonita suit une méthodologie agile[34], ce qui permet de s'adapter rapidement aux demandes de la communauté.
L'aide communautaire de Bonita est donc rapide et de qualité. Le produit obtient une note de 2 sur 2.

Enhydra Shark

L'assistance communautaire d'ES est assez faible. Sur le forum du produit, 18 des 29 questions posées lors des 8 derniers mois sont restées sans réponses. Pour les questions traitées, le délai de réponse est généralement rapide (quelques jours). Les développeurs du produit en sont les principaux auteurs.
Le site web du produit a récemment été modifié. Plusieurs sections du nouveau site sont en cours de construction. L'information y est minimale. De plus, ES ne dispose pas de pages de documentation collaboratives ou de blogs où les utilisateurs peuvent chercher de l'aide. Une note de 1 sur 2 est attribuée à l'assistance communautaire.

6.13.6 Assistance professionnelle

jBPM

Au même titre que Drools, jBPM est un projet de JBoss Inc., filiale du groupe Red Hat.
JBoss offre des services de consultation et une assistance professionnel ainsi que des cours de formation. Plusieurs formules existent avec des niveaux d'assistance différents. La filiale JBoss et le groupe Red Hat en général affichent une forte présence et une

34. http ://www.youtube.com/user/BonitaSoft#p/c/BBDB37F4E5A1D28D/4/I4ekfM-0I90 (dernière consultation le 09-11-10)

bonne réputation dans les milieux d'entreprise. Red Hat est notamment connu pour son service d'assistance de qualité de la distribution Linux Red Hat. La société est cotée en bourse et a présenté des résultats très positifs ces dernières années.
Ainsi, jBPM est financé par une entreprise solide et pérenne, ayant une grande expertise dans le développement de logiciels à code ouvert pour les entreprises.
L'assistance professionnelle de jBPM obtient une note de 2 sur 2.

Bonita Open Solution

La compagnie BonitaSoft a été créée en juin 2009. Elle est financée à hauteur de 3 millions de dollars US et compte des dizaines de clients provenant de différents secteurs dont le secteur bancaire, le secteur public et les assurances [35]. L'offre commerciale possède son propre domaine *bonitasoft.com* alors que le site *bonitasoft.org* est destiné à la communauté.
BonitaSoft fournit des services qui couvrent la conception, le développement, le déploiement et l'utilisation des applications de gestion de processus. Le centre d'assistance technique fournit notamment l'aide au développement et à la production, les patchs et l'accès à un portail client. Selon la formule choisie, le client aura accès à une version certifiée, des mises à jour automatiques, des temps de réponse et de diagnostics garantis, la prise en charge d'un nombre d'incidents illimité, l'assistance par courriel et l'assistance téléphonique.
De plus, les clients peuvent suivre des sessions de formation dans leurs entreprises ou dans les locaux de Bonita. Des conférences sont également organisées à travers le monde.
Enfin, quatre certifications officielles sont disponibles :
- Certification métier BonitaSoft.
- Certification développeur BonitaSoft.
- Certification développeur avancé BonitaSoft.
- Certification administrateur BonitaSoft.

35. http ://www.bonitasoft.com/products/discover.php (dernière consultation le 09-11-10)

Les clients ont ainsi le choix pour une large palette de services payants. Ces services couvrent à la fois la formation au produit et l'assistance lors de son utilisation. Une note de 2 sur 2 est accordée à l'assistance professionnelle.

Enhydra Shark

Enhydra Shark est une marque déposé de la société autrichienne Together Teamlösungen EDV-Dienstleistungen GmbH. La société possède également des bureaux en Slovakie et en Thaïlande et propose d'autres outils de gestion de documents, d'extraction-transformation-chargement de données ainsi que des outils pour MS Exchange, entre autres. La licence commerciale Together Software License V2 donne à ses détenteurs le droit d'installer et d'utiliser un nombre limité de copies du produit. Elle offre également le droit d'utiliser le produit dans des logiciels à code fermé.
L'assistance professionnelle fournit une garantie de bon fonctionnement pour 90 jours. Une garantie des services de maintenance assure l'aide des ingénieurs de la société en cas de problèmes, selon les termes du contrat.
Une licence pour un poste de travail avec une intervention de maintenance est facturée à 999 euros (environs 1 328 CAD). Deux licences et une intervention de maintenance coûtent 1 799 euros (environs 2 390 CAD). Une licence de site avec 5 interventions de maintenance coûtent 7 999 euros (environs 10 630 CAD).
Des services payants de développement et de formation sont également disponibles.
La société Together Teamlösungen EDV-Dienstleistungen GmbH revendique quelques clients sur son site, dont une compagnie suisse qui utilise une version personnalisée de l'éditeur *TWE*.
Together Teamlösungen a moins de visibilité et d'impact sur la communauté des logiciels à code ouvert que Red Hat par exemple (qui finance le projet jBPM). La société affiche également un nombre restreint de clients, dont l'importance est moindre que celle des clients de Bonita (banques, assurances, secteur public...). De plus, l'offre d'assistance professionnelle n'est pas clairement expliquée sur le site du produit. Une note de 1 sur 2 est accordée sur ce critère.

6.14 Résultats des évaluations

6.14.1 Scénario 1

Attribut	Poids	jBPM	BOS	ES
A. Critères fonctionnels				
Langage de définition des processus				
1. Prise en charge de plusieurs moyens d'expression	4	2	2	2
2. Prise en charge de l'internationalisation	4	1	2	1
3. Compréhension du modèle de données	5	2	2	2
Contrôle du flux de travail				
4. Contrôle élémentaire du flux	5	2	2	2
5. Contrôle avancé du flux	5	1	1	2
6. Contrôle selon l'état du processus	4	1	1	1
7. Itération dans le flux	4	1	2	1
8. Annulation d'éléments dans le flux	3	1	1	1
9. Gestion d'instances multiples	3	1	2	1
10. Démarrage déclenché par des signaux externes	5	1	2	1
Distribution des items de travail				
11. Distribution à la conception	3	2	2	2
12. Allocation des ressources par le système de gestion du flux	5	1	1	1
13. Engagement des ressources à exécuter des tâches	4	2	2	2
14. Modification des attributions de tâches aux ressources	3	1	1	1
Représentation et utilisation des données dans le flux				
15. Définition des données pour un flux	4	1	1	1
16. Transfert de données entre composants d'un processus	4	1	2	1
17. Transfert de données entre processus et environnement	5	1	2	1
18. Prise en charge de divers mécanismes de transfert	5	1	2	1
19. Utilisation des données comme conditions	3	2	2	2
Accessibilité aux experts métier				

20. Définition graphique des processus	4	2	2	2
21. Définition assistée des activités	4	1	2	2
22. Interfaces pour non informaticiens	4	2	2	2
23. Abstraction des détails d'implémentation	4	1	2	2
24. Modèles de processus	4	1	1	2
Le moteur de flux de travail				
25. Intégration à l'environnement	5	2	2	2
26. Changements dynamiques	3	2	1	1
Outils de vérification				
27. Vérification de la syntaxe	4	2	2	2
28. Test	3	2	2	1
29. Débogage	3	2	1	2
30. Journalisation et audit	4	2	2	2
Stockage des processus				
31. Versionnage	3	2	2	2
32. Importation/Exportation de processus	5	0	1	2
B. Critères non fonctionnels				
33. Conformité aux standards	5	1	2	2
Performance et coût en ressources				
34. Performance des moteurs de flux	3	1	1	1
35. Coûts en ressources matérielles	3	1	1	1
Prise en main				
36. Documentation	3	2	2	2
37. Convivialité	3	1	2	2
38. Courbe d'apprentissage	4	1	2	2
Évaluation sur l'aspect logiciel libre				
39. Communauté	4	2	2	1
40. Publications	4	2	2	2

41. Longévité	4	1	2	2
42. Licence	5	2	2	2
43. Assistance communautaire	4	2	2	1
44. Assistance professionnelle	3	2	2	1
Totaux pour le scénario 1				
Totaux	346	248	301	274
Pourcentage	100	72	87	79

Tableau 6.2: Évaluation des moteurs de flux selon S1

6.14.2 Scénario 2

Attribut	Poids	jBPM	BOS	ES
A. Critères fonctionnels				
Langage de définition des processus				
1. Prise en charge de plusieurs moyens d'expression	4	2	2	2
2. Prise en charge de l'internationalisation	5	1	2	1
3. Compréhension du modèle de données	5	2	2	2
Contrôle du flux de travail				
4. Contrôle élémentaire du flux	5	2	2	2
5. Contrôle avancé du flux	5	1	1	2
6. Contrôle selon l'état du processus	5	1	1	1
7. Itération dans le flux	4	1	2	1
8. Annulation d'éléments dans le flux	5	1	1	1
9. Gestion d'instances multiples	3	1	2	1
10. Démarrage déclenché par des signaux externes	4	1	2	1
Distribution des items de travail				

11. Distribution à la conception	5	2	2	2
12. Allocation des ressources par le système de gestion du flux	5	1	1	1
13. Engagement des ressources à exécuter des tâches	5	2	2	2
14. Modification des attributions de tâches aux ressources	5	1	1	1
Représentation et utilisation des données dans le flux				
15. Définition des données pour un flux	5	1	1	1
16. Transfert de données entre composants d'un processus	4	1	2	1
17. Transfert de données entre processus et environnement	3	1	2	1
18. Prise en charge de divers mécanismes de transfert	4	1	2	1
19. Utilisation des données comme conditions	3	2	2	2
Accessibilité aux experts métier				
20. Définition graphique des processus	4	2	2	2
21. Définition assistée des activités	4	1	2	2
22. Interfaces pour non informaticiens	4	2	2	2
23. Abstraction des détails d'implémentation	4	1	2	2
24. Modèles de processus	4	1	1	2
Le moteur de flux de travail				
25. Intégration à l'environnement	4	2	2	2
26. Changements dynamiques	5	2	1	1
Outils de vérification				
27. Vérification de la syntaxe	4	2	2	2
28. Test	3	2	2	1
29. Débogage	3	2	1	2
30. Journalisation et audit	4	2	2	2
Stockage des processus				
31. Versionnage	5	2	2	2
32. Importation/Exportation de processus	4	0	1	2
B. Critères non fonctionnels				

33. Conformité aux standards	4	1	2	2
Performance et coût				
34. Performance des moteurs de flux	3	1	1	1
35. Coûts en ressources matérielles	3	1	1	1
Prise en main				
36. Documentation	3	2	2	2
37. Convivialité	3	1	2	2
38. Courbe d'apprentissage	4	1	2	2
Évaluation sur l'aspect logiciel libre				
39. Communauté	4	2	2	1
40. Publications	4	2	2	2
41. Longévité	4	1	2	2
42. Licence	5	2	2	2
43. Assistance communautaire	4	2	2	1
44. Assistance professionnelle	3	2	2	1
Totaux pour le scénario 2				
Totaux	360	262	308	283
Pourcentage	100	73	86	79

Tableau 6.3: Évaluation des moteurs de flux selon S2

6.14.3 Scénario 3

Attribut	Poids	jBPM	BOS	ES
A. Critères fonctionnels				
Langage de définition des processus				
1. Prise en charge de plusieurs moyens d'expression	4	2	2	2

2. Prise en charge de l'internationalisation	3	1	2	1
3. Compréhension du modèle de données	5	2	2	2
Contrôle du flux de travail				
4. Contrôle élémentaire du flux	5	2	2	2
5. Contrôle avancé du flux	5	1	1	2
6. Contrôle selon l'état du processus	5	1	1	1
7. Itération dans le flux	5	1	2	1
8. Annulation d'éléments dans le flux	4	1	1	1
9. Gestion d'instances multiples	5	1	2	1
10. Démarrage déclenché par des signaux externes	4	1	2	1
Distribution des items de travail				
11. Distribution à la conception	4	2	2	2
12. Allocation des ressources par le système de gestion du flux	5	1	1	1
13. Engagement des ressources à exécuter des tâches	3	2	2	2
14. Modification des attributions de tâches aux ressources	4	1	1	1
Représentation et utilisation des données dans le flux				
15. Définition des données pour un flux	4	1	1	1
16. Transfert de données entre composants d'un processus	5	1	2	1
17. Transfert de données entre processus et environnement	4	1	2	1
18. Prise en charge de divers mécanismes de transfert	3	1	2	1
19. Utilisation des données comme conditions	4	2	2	2
Accessibilité aux experts métier				
20. Définition graphique des processus	3	2	2	2
21. Définition assistée des activités	3	1	2	2
22. Interfaces pour non informaticiens	3	2	2	2
23. Abstraction des détails d'implémentation	3	1	2	2
24. Modèles de processus	5	1	1	2
Le moteur de flux de travail				

25. Intégration à l'environnement	4	2	2	2
26. Changements dynamiques	4	2	1	1
Outils de vérification				
27. Vérification de la syntaxe	5	2	2	2
28. Test	5	2	2	1
29. Débogage	5	2	1	2
30. Journalisation et audit	4	2	2	2
Stockage des processus				
31. Versionnage	4	2	2	2
32. Importation/Exportation de processus	3	0	1	2
B. Critères non fonctionnels				
33. Conformité aux standards	4	1	2	2
Performance et coût				
34. Performance des moteurs de flux	4	1	1	1
35. Coûts en ressources matérielles	3	1	1	1
Prise en main				
36. Documentation	3	2	2	2
37. Convivialité	3	1	2	2
38. Courbe d'apprentissage	4	1	2	2
Évaluation sur l'aspect logiciel libre				
39. Communauté	4	2	2	1
40. Publications	4	2	2	2
41. Longévité	4	1	2	2
42. Licence	5	2	2	2
43. Assistance communautaire	3	2	2	1
44. Assistance professionnelle	2	2	2	1
Totaux pour le scénario 3				
Totaux	350	255	299	274

Pourcentage	100	73	85	78

Tableau 6.4: Évaluation des moteurs de flux selon S3

6.14.4 Scénario 4

Attribut	Poids	jBPM	BOS	ES
A. Critères fonctionnels				
Langage de définition des processus				
1. Prise en charge de plusieurs moyens d'expression	3	2	2	2
2. Prise en charge de l'internationalisation	3	1	2	1
3. Compréhension du modèle de données	5	2	2	2
Contrôle du flux de travail				
4. Contrôle élémentaire du flux	5	2	2	2
5. Contrôle avancé du flux	5	1	1	2
6. Contrôle selon l'état du processus	3	1	1	1
7. Itération dans le flux	4	1	2	1
8. Annulation d'éléments dans le flux	3	1	1	1
9. Gestion d'instances multiples	3	1	2	1
10. Démarrage déclenché par des signaux externes	3	1	2	1
Distribution des items de travail				
11. Distribution à la conception	3	2	2	2
12. Allocation des ressources par le système de gestion du flux	5	1	1	1
13. Engagement des ressources à exécuter des tâches	3	2	2	2
14. Modification des attributions de tâches aux ressources	3	1	1	1
Représentation et utilisation des données dans le flux				
15. Définition des données pour un flux	3	1	1	1

16. Transfert de données entre composants d'un processus	3	1	2	1
17. Transfert de données entre processus et environnement	3	1	2	1
18. Prise en charge de divers mécanismes de transfert	3	1	2	1
19. Utilisation des données comme conditions	3	2	2	2
Accessibilité aux experts métier				
20. Définition graphique des processus	5	2	2	2
21. Définition assistée des activités	5	1	2	2
22. Interfaces pour non informaticiens	5	2	2	2
23. Abstraction des détails d'implémentation	5	1	2	2
24. Modèles de processus	5	1	1	2
Le moteur de flux de travail				
25. Intégration à l'environnement	4	2	2	2
26. Changements dynamiques	3	2	1	1
Outils de vérification				
27. Vérification de la syntaxe	3	2	2	2
28. Test	3	2	2	1
29. Débogage	3	2	1	2
30. Journalisation et audit	4	2	2	2
Stockage des processus				
31. Versionnage	3	2	2	2
32. Importation/Exportation de processus	3	0	1	2
B. Critères non fonctionnels				
33. Conformité aux standards	4	1	2	2
Performance et coût				
34. Performance des moteurs de flux	3	1	1	1
35. Coûts en ressources matérielles	5	1	1	1
Prise en main				
36. Documentation	5	2	2	2

37. Convivialité	5	1	2	2
38. Courbe d'apprentissage	5	1	2	2
Évaluation sur l'aspect logiciel libre				
39. Communauté	5	2	2	1
40. Publications	4	2	2	2
41. Longévité	4	1	2	2
42. Licence	5	2	2	2
43. Assistance communautaire	5	2	2	1
44. Assistance professionnelle	5	2	2	1
Totaux pour le scénario 4				
Totaux	344	253	300	276
Pourcentage	100	73	87	80

Tableau 6.5: Évaluation des moteurs de flux selon S4

6.14.5 Bilan

On constate que Bonita Open Solution arrive en tête suivi d'Enhydra Shark puis de jBPM pour les quatres scénarios. Contrairement aux moteurs de règles, les résultat des moteurs de flux de travail évalués sont plus proches les uns des autres. L'écart maximal obtenu est de 15% entre les notes de jBPM et de Bonita Open Solution au scénario 1, contre un écart de 28% entre Drools et Clips pour le scénario 4 d'évaluation des moteurs de règles. Globalement, les trois moteurs de flux répondent aux exigences. La plus mauvaise note obtenue est au dessus de 70% de satisfaction. Par ailleurs, contrairement à Drools, aucun moteur de flux de travail n'a franchi la barre des 90% de satisfaction. Ceci est principalement dû au fait que plusieurs patrons de contrôle avancé de flux, d'allocation des ressources et de représentation des données ne sont pas directement

exprimables par les langages de définition de processus des différents produits.

jBPM obtient sa plus mauvaise note pour le scénario impliquant la collaboration avec l'environnement externe. Ceci s'explique notamment par l'absence de mécanismes d'importation et d'exportation des définitions de processus et par la faible conformité aux standards établis.

Enhydra Shark obtient sa meilleure évaluation (80%) lorsque l'organisme dispose d'un nombre limité d'informaticiens. Ceci est dû à la qualité et à l'accessibilité des outils graphiques de définition et d'administration des processus.

Pour les quatres scénarios, l'avance de Bonita Open Solution s'explique par sa prise en charge directe de l'instanciation multiple, de l'itération dans le flux et du transfert de données via des messages *Throw* et *Catch*. Les deux produits concurrents ne proposent pas d'équivalents et la réalisation de ces fonctionnalités nécessite un effort de développement.

Ce chapitre nous a permis d'évaluer les moteurs de flux de travail jBPM, Bonita Open Solution et Enhydra Shark selon les critères établis au Chapitre IV. Les résultats ont été pondérés pour quatre scénarios représentant quatre environnements types où les produits peuvent être utilisés.

Le chapitre suivant clos l'évaluation en produisant une analyse et des recommandations détaillées pour les moteurs de règles et les moteurs de flux de travail étudiés.

CHAPITRE VII

RECOMMANDATIONS

7.1 Points forts et points faibles des produits évalués

7.1.1 Moteur de règles

Drools permet d'exprimer les règles à l'aide de plusieurs langages très distincts dans leurs approches (Java, Scala, CLIPS, tableurs...). La définition de DSLs apporte de l'expressivité. L'internationalisation est aisément réalisable avec les bibliothèques i18n de Java et le modèle de données est directement accessible à partir des règles.
Drools prend partiellement en charge les modèles de règles, mais satisfait pleinement l'invocation de procédures externes et l'utilisation de tables décisionnelles.
Le système est accessible aux utilisateurs métier tout en offrant des outils avancés aux développeurs pour le test, le débogage, la journalisation et l'audit. L'interface Guvnor et le plugiciel *eclipse* sont notamment très soignés et ergonomiques. Par ailleurs, Drools Flow est un module de gestion de flux de règles disposant de fonctionnalités avancées.
Drools satisfait partiellement l'explication de raisonnement et le changement dynamique des règles. La définition de méta-données et la recherche par critères sont également perfectibles. Enfin, les performances générales et le coût en ressources matérielles sont plutôt moyens.
Une documentation de qualité, s'adressant aux utilisateurs de tous les niveaux est fournie avec le produit. Enfin, les critères relatifs à l'aspect logiciel libre sont pleinement satisfaits.

CLIPS offre un choix limité de moyens d'expression des règles et ne prend pas en charge l'internationalisation. Le langage est assez expressif et les possibilités d'accès au modèle de données sont honorables.
CLIPS permet d'invoquer des procédures externes et de définir des flux de règles mais ne gère pas les modèles de règles, ni les tables décisionnelles.
Le système est assez peu accessible aux utilisateurs métier et ne propose pas de dépôts de règles. Par contre, le moteur d'inférence est très performant et possède des fonctionnalités avancées tel que le changement dynamique de règles, le maintient de la vérité et l'émulation du raisonnement par chaînage arrière.
Des outils de vérification sont mis à disposition des informaticiens, dont la coloration syntaxique, le débogage et le test. À moindre degré, la résolution des conflits, la journalisation et l'audit sont possibles avec CLIPS.
La système jouit d'une documentation de qualité, d'une communauté nombreuse et active, d'une longévité remarquable et d'une licence en accord avec les exigences de la chaire.

OpenRules ne fournit qu'un seul moyen d'exprimer les règles (code Java dans des tables décisionnelles). L'utilisation de Java permet de profiter des bibliothèques i18n d'internationalisation et d'accéder à différentes sources de données externes.
Le système peut invoquer des procédures Java dans les règles et offre des fonctionnalités limitées de flux et de modèles de règles.
Grâce aux tables décisionnelles et à la séparation entre code applicatif et logique métier, OpenRules est accessible aux non informaticiens.
La notion de moteur d'inférence n'existe pas pour le moteur de règles qui ne fait qu'exécuter simplement une suite d'instructions Java.
Le produit ne fait que réutiliser les outils de vérification de l'écosystème Java. Contrairement à la concurrence, il n'offre pas de commandes de débogage propres aux règles, ni d'audits sophistiqués comme Drools.
Plusieurs types de dépôts sont proposés par OpenRules. Toutefois, les fonctions de recherche par critères, de versionnage et de définition de méta-données sont limitées.

La documentation fournie est de qualité. Le produit dispose d'une longévité respectable et de publications fréquentes. Par contre, il n'existe pas de communauté et les contacts se font uniquement avec l'entreprise. La licence utilisée présente quelques restrictions. L'assistance gratuite est limité et l'assistance payante est, à notre sens, onéreuse.

Comme le montrent les résultats obtenus pour les quatre environnements types qu'on a défini, Drools est recommandé quelque soit le profil de l'organisme financier. En effet, il allie le choix à l'expressivité, un moteur d'inférence aux fonctionnalités et aux performances honorables à une haute accessibilité pour les utilisateurs métier. L'intégration aux systèmes existants dans les organismes devrait être aisée. De plus, la courbe d'apprentissage faible et la documentation de qualité facilitent l'adoption du produit.

CLIPS serait recommandé pour les organismes ayant des exigences élevées en matière de performances et disposant d'une importante équipe de développeurs. En effet, le moteur d'inférence de la dernière version de CLIPS est plus rapide que les moteurs commerciaux les plus avancés, et ce pour quasiment tous les bancs d'essais. Le système ne bénéficie toutefois pas d'une encapsulation pour les utilisateurs métier, d'où le besoin d'avoir des informaticiens à temps plein dans l'organisme de finance sociale et solidaire.

Quant à OpenRules, il est surtout recommandé pour les organismes dont une large partie des données métier sont enregistrées dans des tables décisionnelles. Le produit donne la main aux experts métier pour concevoir et maintenir une large partie du système. Il est donc judicieux d'opter pour ce troisième produit dans les organismes où les experts métier sont enthousiastes à contribuer au système informatique et où l'équipe d'informaticiens est réduite.

7.1.2 Moteur de flux

jBPM, Bonita Open Solution et Enhydra Shark permettent d'exprimer les processus et les activités qui les composent selon plusieurs langages. Il est possible de manipuler des entités du modèle de données à partir des trois produits. Les patrons de contrôles élémentaires de flux, de distribution des tâches à la conception et de définition

de préconditions et de postconditions sont unanimement pris en charge.

Au niveau de l'accessibilité aux experts métier, les trois systèmes permettent de définir graphiquement les processus et proposent des interfaces pour non informaticiens. Les moteurs de flux de travail s'intègrent à des environnements existants et les couches présentation, métier et données peuvent être configurées pour adhérer à une architecture existante. Les critères de vérification de la syntaxe, de journalisation des événements et de versionnage des processus sont totalement satisfaits pour les trois produits. Pour ce qui est des critères non fonctionnels, les systèmes évalués présentent des documentations de qualité et destinées à la fois aux informaticiens et aux non informaticiens. Les fréquences de publication des trois concurrents sont élevées et leurs licences sont conformes aux exigences de la Chaire.

jBPM se distingue par une prise en charge avancée des changements dynamiques. Le système permet de migrer des instances de processus en cours d'exécution vers de nouvelles versions. L'utilisateur configure la migration en effectuant des correspondances entre les éléments des versions source et destination des processus. jBPM dispose également de ses propres outils de débogage. Il est possible de suivre en temps réel l'exécution des processus ainsi que l'évolution des valeurs de leurs variables. Le produit est désavantagé par l'impossibilité d'exprimer plusieurs patrons de contrôle de flux, de distribution de tâches et de représentation des données. En effet, la flexibilité de la machine d'exécution de processus PVM est un atout mais le système devrait proposer des implémentations pour les patrons établis tout en gardant la possibilité de les remplacer par des implémentations personnelles. Par ailleurs, l'utilisation de jBPM se fait souvent avec du code Java. Le produit présente donc une faible abstraction des détails d'implémentation et les assistants fournis par les interfaces graphiques du produit ne permettent de réaliser qu'un nombre limité d'opérations. Il en résulte une forte courbe d'apprentissage et une difficulté d'accès aux experts métier. De plus, jBPM souffre d'une faible conformité aux standards. Il n'est pas possible d'importer ou d'exporter des définitions de processus sous des formats autres que le format propriétaire jBPM ou une implémentation très incomplète de la norme BPMN. Enfin, les deux développeurs principaux du produit ont quitté le projet et lancé leur propre système de gestion de

flux de travail. La pérennité de jBPM est donc menacée.

Bonita Open Solution et Enhydra bénéficient d'une large palette d'assistants guidant l'utilisateur dans la création des agents automatiques, la modification des propriétés des éléments de processus ou encore l'administration du système. Les éditeurs fournis permettent d'abstraire les détails d'implémentation. Il est possible de définir les processus et leurs activités en manipulant très peu de code. Il en résulte une courbe d'apprentissage faible et une convivialité accrue. Les deux produits présentent une bonne conformité aux standards. Bonita Open Solution utilise une représentation graphique conforme à la norme BPMN 2.0 et Enhydra Shark se base sur les spécifications WfMC et la norme XPDL 1.0.

Bonita Open Solution se distingue par des assistants facilitant l'internationalisation. L'architecture des connecteurs intègre la prise en charge de la localisation. Les libellés des éléments de processus peuvent être écrits dans des caractères non ASCII et les interfaces de l'éditeur et de l'application web User XP sont disponibles en plusieurs langues. Un second atout de BOS réside dans la prise en charge directe de l'itération. Des filtres et des connecteurs prédéfinis permettent de définir des scénarios avancés d'instanciation multiples d'activités. Par ailleurs, l'utilisation de messages permet d'exprimer directement le déclenchement d'activités suite à la réception de signaux externes. Les données peuvent également transiter entre composants de processus et avec l'environnement externe grâce aux messages. Contrairement à ses concurrents, Bonita Open Solution ne propose pas ses propres outils de débogage.

Concernant Enhydra Shark, le produit est le seul à proposer des implémentations des patrons de contrôle de flux avancés. Il se distingue également par la gestion des modèles de processus. L'inter-opérabilité est un autre point fort : ES permet d'importer et d'exporter des définitions de processus à partir de tout moteur de flux conforme au format XPDL 1.0. De plus, le produit implémente l'interface 4 du modèle de référence WfMC, permettant les communications inter-moteurs de flux. Au rayon des points faibles, Enhydra Shark ne fournit pas ses propres classes et outils de test, la communauté autour du produit est peu nombreuse et peu active, l'assistance gratuite est lente et Together Teamlösungen, société détentrice du produit, ne dispose pas du même niveau de

ressources que ses concurrents.

Comme l'indiquent les résultats des évaluations, Bonita Open Solution est recommandable pour les quatres environnements types définis. Il doit son avance à la prise en charge des patrons de données, à son accessibilité aux experts métier, à sa prise en main facile et à la qualité de sa communauté.

jBPM est recommandable lorsque l'organisme client a besoin d'une flexibilité maximale et qu'il dispose d'une équipe de développeurs prêts à fournir des efforts de développement conséquents. Ces développeurs bénéficieront d'outils de test et de débogage avancés, fournis par le produit. jBPM est également recommandé lorsque les définitions de processus changent rapidement et qu'il est nécessaire de migrer les instances en cours d'exécution vers les nouvelles versions.

Enhydra Shark est conseillé principalement pour les organismes qui veulent utiliser un système conforme aux spécifications de la WfMC et au format XPDL. Des trois produits évalués, Enhydra Shark est probablement le moteur de flux le plus rigoureux dans sa conformité aux standards établis. Au niveau de la convivialité et de l'accessibilité aux experts métiers, ES est d'aussi bonne qualité que BOS.

7.2 Solution recommandée pour la gestion de règles et de processus d'affaires

Le système de gestion de règles d'affaires Drools et le système de gestion de flux de travail Bonita Open Solution arrivent en tête dans leurs catégories respectives pour tous les environnements types définis. On recommande donc Drools pour la gestion des règles d'affaires et Bonita Open Solution pour la gestion de flux de travail.

7.2.1 Apports de JBoss Drools

La gestion des règles d'affaires par Drools apportera les bénéfices suivants aux organismes de finance sociale et solidaire :

- Adaptation au changement : Découpler les règles d'affaires du code applicatif facilite

leur maintenance et leur réutilisation.

- Accessibilité aux experts métier : L'ergonomie du plugiciel *eclipse*, l'application web Guvnor et les assistants de création de règles Drools permettent aux experts métier de gérer les règles d'affaires en limitant le recours aux services d'informaticiens. Guvnor dispose d'un mécanisme d'authentification des utilisateurs et de gestion des permissions qui permet de contrôler l'accès des utilisateurs au système et de tracer les modifications qu'ils ont effectué.
- Expressivité du langage : Le style déclaratif adopté pour la définition de règles Drools, combiné à la possibilité de définir des langages spécifiques au domaine d'affaires (DSLs) améliorent la lisibilité des règles en permettant aux utilisateurs d'exprimer facilement et succinctement leurs spécifications.
- Amélioration des performances : L'appariement de modèles (pattern matching) et le partage de conditions utilisés par le moteur d'inférence de Drools permettent d'appliquer des ensembles de règles complexes à un volume important de faits. Les temps d'exécution obtenus sont meilleurs qu'avec l'utilisation de conditions imbriquées d'un langage impératif tel que Java.
- Gestion des dépôts de règles : Les règles peuvent être stockées dans des dépôts. Il est possible de les hiérarchiser et de leur affecter des numéros de versions. L'administrateur des dépôts peut facilement basculer entre les versions, répliquer les dépôts de règles ou encore effectuer des recherches par critères.
- Vérification des règles : Drools fournit des outils de vérification syntaxique, de résolution de conflits, de débogage, de test, de journalisation et d'audit des règles. Ces fonctionnalités permettent aux développeurs de s'assurer de la validité des règles et de rapidement détecter et corriger les éventuelles erreurs.
- Avantages du logiciel libre : Drools utilise la licence libre Apache, qui permet aux organismes de finance sociale et solidaire de consulter, modifier et reproduire le code source du produit en toute indépendance d'un quelconque fournisseur privé de logiciels. Les organismes peuvent bénéficier gratuitement de l'assistance de la communauté Drools sur les forums publiques. La plate-forme de signalisation et de suivi de bogues est également gratuitement disponible. S'ils le désirent, les clients peuvent acquérir des

services d'assistance payants, assurés par la société Red Hat.

7.2.2 Apports de Bonita Open Solution

La gestion des flux de travail par Bonita Open Solution permettra aux organismes de finance sociale et solidaire de :

- Rationaliser leurs processus d'affaires : La formalisation des processus assure que toutes les étapes sont correctement suivies. L'uniformité des définitions garantit une exécution identique pour toutes les instances d'un même processus.
- Augmenter l'efficacité : L'automatisation de plusieurs processus d'affaires induit l'élimination d'étapes non nécessaires et accélère l'avancement des flux de travail. Les participants aux processus peuvent simultanément exécuter des tâches. L'acheminement manuel de documents papier est réduit.
- Disposer d'informations en temps réel : Il est possible de connaître instantanément l'état des processus. Les gestionnaires peuvent alors prendre des décisions plus précises, étant donné qu'ils connaissent la situation actuelle exacte des processus. À partir de l'application User XP de Bonita, un utilisateur est capable de visualiser l'état des processus actifs et les valeurs actuelles des données échangées.
- Évaluer l'efficacité des processus et des employés : L'analyse des processus amène à leur remaniement en concordance avec les besoins en perpétuel changement. Les audits sur les journaux d'exécution permettent de détecter les goulots d'étranglement dans le flux, le niveau d'efficacité de chaque employé ou encore les données inutiles ou manquantes dans la réalisation des processus. À cet effet, Bonita permet de configurer les différents aspects de journalisation, fournit des journaux dédiés au moteur de flux, persiste les historiques d'exécution dans une base de données et propose des rapports d'audit via l'application User XP.
- Exprimer facilement des processus sophistiqués : Bonita dispose d'assistants permettant entre autres de répéter une même activité ou d'en créer plusieurs instances. Le produit permet de modéliser les transferts de données entre les composants d'un processus et avec l'environnement externe.

- Définir et administrer les processus d'affaires par les experts métier : L'éditeur graphique intuitif et puissant de Bonita facilite la définition de processus et de formulaires de communication avec les participants humains. L'interface User XP de Bonita permet aux utilisateurs de visualiser et d'exécuter leurs tâches. L'application instaure un système d'authentification et de gestion des permissions afin de contrôler l'accès aux fonctions d'administration du système.
- Profiter des atouts d'un produit libre : La licence GPLv2 utilisée par Bonita permet aux organismes d'avoir un contrôle total sur le fonctionnement du logiciel. De nombreux canaux de communication permettent d'instaurer un échange constant avec les développeurs et procurent une réelle proximité. Les utilisateurs sont appelés a participer à tous les niveaux et leur avis influence l'évolution du produit. Par ailleurs, l'assistance communautaire de Bonita est rapide et de qualité, grâce à une grande implication des développeurs et à une communauté nombreuse.

7.2.3 Collaboration entre les deux systèmes

Comme nous l'avons vu au chapitre VI, BOS dispose d'un connecteur prédéfini permettant d'invoquer le moteur de règles Drools. Un ensemble de faits, éventuellement tirés du modèle de données global du système informatique, est passé au connecteur à partir de BOS. L'ensemble de règles d'affaires Drools à exécuter sur ces faits est également indiqué au connecteur. Ce dernier démarre une session Drools sans stockage d'état (stateless) et retourne les valeurs obtenues suite à l'exécution des règles au processus Bonita appelant. Des correspondances peuvent être établies entre les valeurs retournées et des variables de processus.

Grâce à cet échange, il est possible de déléguer l'évaluation de règles complexes au moteur de règles et d'utiliser les résultats obtenus pour orienter le flux de travail. Les règles d'affaires peuvent être stockées et hiérarchisées dans des dépôts Guvnor et réutilisées dans plusieurs définitions de processus BOS. La maintenance du système s'en trouve facilitée, vu que la modification d'une règle est répercutée sur tous les processus y faisant appel.

La collaboration entre Drools et BOS peut se faire dans l'autre sens. Lors de son exécution, Drools pourrait atteindre un point où des données additionnelles sont requises dans le processus de décision. En utilisant l'API Java de BOS, il déclencherait une instance de processus pour alerter les participants concernés et leur faire exécuter les tâches requises afin de récolter les données manquantes. Lorsque l'exécution de l'instance de processus est terminée, les résultats sont récupérés par Drools et ajoutés à la mémoire de travail.

CONCLUSION

La Chaire ambitionne de poser les fondements d'une famille de logiciels libres pour la finance sociale et solidaire. Ce mémoire vise à recommander à la chaire des solutions libres et de qualité pour la gestion des règles d'affaires et des flux de travail. En effet, les moteurs de règles permettent aux organismes financiers de découpler leurs règles d'affaires de leur code applicatif et de faciliter leur maintenance et leur évolution. Les moteurs de flux de travail rationalisent les processus et améliorent la productivité, entre autres avantages.

Pour produire des recommandations fiables, on a d'abord explicité la signification du terme *logiciel libre* ainsi que les concepts majeurs et l'état de l'art pour les moteurs de règles et de flux de travail. Troisièmement, nous avons établi des critères d'évaluation pour les deux types de systèmes. Ces critères sont à la fois des exigences techniques recensées dans la littérature mais également des exigences propres à la finance sociale et solidaire. En effet, nous avons insisté sur les besoins spécifiques à notre contexte, dont l'accessibilité des produits aux experts métier, la prise en charge de l'internationalisation, l'adéquation de la licence utilisée aux besoins de la chaire, l'importance de la communauté autour du logiciel ou encore la facilité de prise en main du système. Les critères établis ont été pondérés pour plusieurs environnements types, illustrant différents profils d'organismes de finance sociale et solidaire.

Munis de nos grilles d'évaluation, nous avons ensuite analysé trois systèmes de gestion de règles, à savoir JBoss Drools, CLIPS et OpenRules. De même, les trois systèmes de gestion de flux de travail JBoss jBPM, Bonita Open Solution et Enhydra Shark ont été notés selon les critères précédemment établis.

À la lumière de ces évaluations, nous avons été capable de recommander Bonita Open Solution pour la gestion des flux de travail et JBoss Drools pour celle des règles d'affaires.

Ces deux produits peuvent coopérer et répondent le mieux aux exigences définies.

Ce travail fût l'occasion de comprendre le fonctionnement des moteurs de règles et des moteurs de flux. Ce fût également une opportunité pour apprendre à rechercher et à analyser des publications scientifiques. De plus, l'évaluation de plusieurs alternatives selon une grille de critères et un système de notation est une approche efficace, applicable dans plusieurs contextes et qui permet d'obtenir des évaluations objectives.

Si les moteurs de règles et de flux de travail choisis correspondent le mieux aux besoins de la chaire, il se peut que leur enrichissement avec des fonctionnalités additionnelles améliore leur adéquation aux besoins. Dans le cadre de futurs travaux, il serait utile de développer des modules supplémentaires propres au domaine de la finance, qui seraient greffés aux produits sélectionnés. Ceci est notamment possible grâce aux licences de Drools et de Bonita Open Solution qui permettent aux utilisateurs de les modifier et de les réutiliser. De plus, Drools permet de définir des langages spécifiques à un domaine, qui est le domaine financier dans le cadre de la chaire. Bonita Open Solution offre lui aussi des outils facilitant son extension, notamment la possibilité de définir des connecteurs personnalisés pour interagir avec d'autres systèmes informatiques.

BIBLIOGRAPHIE

Ader, M. 2004. Workflow comparative study. Rapport, Workflow And Reengineering International Association (WARIA).

Bali, M. 2009. *Drools JBoss Rules 5.0 Developer's Guide.* Packt Publishing.

Batory, D. 1994. The leaps algorithms. Rapport, University of Texas at Austin, Austin, TX, USA.

Blow, M., Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, et M. Rowley. 2004. « Bpelj : Bpel for java. ». In *BPELJ : BPEL for Java.* BEA and IBM.

Bolie, J., M. Cardella, S. Blanvalet, et al. 2006. *BPEL Cookbook.* Packt Publishing.

Browne, P. 2009. *JBoss Drools Business Rules.* Packt Publishing.

Chinn, S. J. et G. R. Madey. 1999. « Extending clips to support temporal representation and reasoning », *Expert Systems : The International Journal of Knowledge Engineering and Neural Networks*, vol. 16, no. 2, p. 72–86.

Cirstea, H. et C. Kirchner. 2004. Production systems and rete algorithm formalisation. Rapport, Institut national de recherche en informatique et automatique (INRIA), France.

Coalition, W. M. 1999. Workflow management coalition terminology glossary - wfmc-tc-1011.

Coalition, W. M. 2005. Process definition interface – XML process definition language, version 1.15. Rapport no. WFMC-TC-1025, Workflow Management Coalition.

Coalition, W. M. 2008. Process definition interface – xml process definition language. Specification no. WFMC-TC-1025, Workflow Management Coalition.

Davenport, T. H. 1993. *Process innovation : reengineering work through information technology.* Boston, MA, USA : Harvard Business School Press.

d'Orfeuil, H. R. 2005. « Ethical, solidarity-based citizen involvement in the economy : A prerequisite for social cohesion », *Trends in social cohesion*, no. 12, p. 26–30.

Duijnhouwer, F. et C. Widdows. 2003. Open source maturity model. Rapport, Capgemini.

Fitzgerald, B. 2006. « The transformation of open source software », *Management Information Systems Quarterly*, vol. 30, no. 3, p. 587–598.

Gao, Y. 2006. Bpmn-bpel transformation and round trip engineering.

Garces, R., T. de Jesus, J. Cardoso, et P. Valente. 2009. *Open Source Workflow Management Systems : A Concise Survey*, p. 333–346. Future Strategies Inc.

Gendron, C. et G. L. Bourque. 2003. « Une finance responsable à l'ère de la mondialisation économique », *L'Économie politique*, no. 18, p. 50–61.

Gillmann, M., R. Mindermann, et G. Weikum. 2000. « Benchmarking and configuration of workflow management systems ». In *In Cooperative Information Systems, 7th International Conference, CoopIS 2000*, p. 186–197, Saarbruecken, Germany. University of the Saarland.

Golden, B. 2004. *Succeeding with Open Source*. Addison-Wesley.

Graham, I. 2005. Service oriented business rules management systems. Rapport, Trimere International Ltd.

Group, B. R. 2000. Defining business rules - what are they really ?, final report, revision 1.3.

Homeier, P. V. et T. C. Le. 1991. Eclips : An extended clips for backward chaining and goal-directed reasoning.

Jenz et Partner. 2000a. Software selection criteria schema : Business process modeling tools. Rapport, Jenz and Partner GmbH.

———. 2000b. Software selection criteria schema for business rule tools. Rapport, Jenz and Partner GmbH.

Martin, L. 2008. Projet de chaire de logiciel libre - finance sociale et solidaire.

Morgan, T. 2002. *Business Rules and Information Systems*. Addison-Wesley.

Owen, M. et J. Raj. 2003. Bpmn and business process management : Introduction to the new business process modeling standard.

Pant, K. 2008. *Business Process Driven SOA using BPMN and BPEL : From Business Process Modeling to Orchestration and Service Oriented Architecture*. Packt Publishing.

Russell, N., W. van der Aalst, et A. ter Hofstede. 2006. Exception handling patterns in process-aware information systems. Rapport, BPM Center Report BPM-06-04.

Russell, N., W. van der Aalst, A. ter Hofstede, et D. Edmond. 2004a. Workflow data patterns. Rapport, Queensland University of Technology, Brisbane, USA.

———. 2004b. Workflow resource patterns. Rapport, Eindhoven University of Technol-

ogy, Eindhoven, Netherlands.

Russell, N., W. van der Aalst, A. ter Hofstede, et N. Mulyar. 2006. Workflow control-flow patterns : A revised view. Rapport, BPM Center Report BPM-06-04.

Salatino, M. 2010. *jBPM Developer Guide*. Packt Publishing.

Software, L. 2008. What is the difference between workflow engines and bpm suites ? Rapport, Lombardi Software.

Sottara, D., P. Mello, et M. Proctor. 2008. « Adding uncertainty to a rete-oo inference engine ». In *RuleML '08 : Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*, p. 104–118, Berlin, Heidelberg. Springer-Verlag.

Stallman, R. 2002. *Free Software, Free Society :Selected Essays of Richard M. Stallman*. GNU Press.

van den Berg, K. 2005. « Finding open options ». Mémoire de maîtrise, the Netherlands, Tilburg University.

van der Aalst, W. v. d. et K. v. van Hee. 2002. *Workflow Management : Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press.

Wheeler, D. A. 2007. Why open source software / free software (oss/fs, floss, or foss) ? look at the numbers !

White, S. A. 2003. *XPDL and BPMN*, p. 221–238. Future Strategies Inc.

Wohed, P., B. Andersson, A. H. T. Hofstede, N. Russell, et W. M. V. der Aalst. 2009a. « Patterns-based evaluation of open source bpm systems », *Information and Software Technology*, vol. 51, p. 1187–1216.

Wohed, P., N. Russell, A. H. M. ter Hofstede, B. Andersson, et W. M. P. van der Aalst. 2009b. « Patterns-based evaluation of open source bpm systems : The cases of jbpm, openwfe, and enhydra shark », *Inf. Softw. Technol.*, vol. 51, no. 8, p. 1187–1216.

Wright, I. et J. Marshall. 2003. « The execution kernel of rc++ : Rete*, a faster rete with treat as a special case », *International Journal of Intelligent Games and Simulation*, vol. 2, no. 1, p. 36–48.