

IBM Cúram Social Program Management



Cúram Express Rules Reference Manual

Version 6.05

IBM Cúram Social Program Management



Cúram Express Rules Reference Manual

Version 6.0.5

Important

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations contenues dans la rubrique «Remarques», à la page 261

Dernière révision : mai 2013

Cette édition s'applique à IBM Cúram Social Program Management v6.0 5 et à toutes les versions ultérieures, sauf indication contraire dans de nouvelles éditions.

Eléments sous licence - Propriété d'IBM.

© Copyright IBM Corporation 2012, 2013.

© Cúram Software Limited. 2011. All rights reserved.

Table des matières

Figures	vii	4.4 Gestion des types de données	39
Tableaux	ix	4.4.1 Types de données pris en charge	39
Avis aux lecteurs canadiens.	xi	4.4.2 Où spécifier des types de données	48
Chapitre 1. Introduction du manuel de référence	1	4.5 Gestion des données variables au fil du temps	49
1.1 Objet	1	4.5.1 Définition des données chronologiques	49
1.2 Public concerné	1	4.5.2 Comparaison de la chronologie et de perspectives ponctuelles de temps	52
1.3 Documentation connexe	1	4.5.3 Construction de chronologies	57
1.4 Structure de ce manuel de référence	2	4.5.4 Exploitation des chronologies	62
Chapitre 2. Aperçu de CER	3	4.5.5 Test des chronologies de sortie	68
2.1 Qu'est-ce que CER ?	3	4.5.6 Propriétés chronologiques	74
2.1.1 Langage des règles	3	4.6 Déclenchement de recalcul en cas de changement de données	76
2.1.2 Création et environnement de test	4	Chapitre 5. Gestionnaire de dépendance	77
2.1.3 Environnement d'exécution	5	5.1 Concepts du gestionnaire de dépendance	77
2.2 Quelles sont ses avantages ?	5	5.2 Fonctions du gestionnaire de dépendance	80
2.3 Mode d'utilisation	6	5.2.1 Stockage des enregistrements de dépendance	80
2.4 Principes de base	6	5.2.2 Capture des éléments de changements précédents	83
Chapitre 3. Développement et outils de test CER	9	5.2.3 Identification des éléments dépendants potentiellement affectés	85
3.1 Introduction	9	5.2.4 Recalcul des éléments dépendants identifiés	85
3.2 Editeur de règles CER	9	5.3 Traitement différé du gestionnaire de dépendance	86
3.3 Prise en charge de la localisation	9	5.3.1 Définition de la limite du système pour le traitement différé	86
3.3.1 Localisation des données calculées	9	5.3.2 Traitement des erreurs dans le traitement différé	86
3.3.2 Localisation des descriptions d'artefacts de règles CER	13	5.4 Traitement par lots du gestionnaire de dépendance	87
3.4 Valideur de jeu de règles	13	5.4.1 Soumission de l'ensemble de changements précédents	87
3.5 Interpréteur de jeu de règles	13	5.4.2 Effectuer des recalculs de lots à partir de l'ensemble de changements précédents	88
3.5.1 Lecture du jeu de règles	15	5.4.3 Achèvement de l'ensemble de changements précédents	90
3.5.2 Démarrage d'une session CER	15	5.5 Intégration entre CER et le gestionnaire de dépendance	91
3.5.3 Créer un nouvel objet de règle	15	5.5.1 Utilitaire CER permettant d'identifier les dépendances à stocker	91
3.5.4 Exécution de règles	16	5.5.2 CER utilise le gestionnaire de dépendance pour stocker les dépendances liées à des valeurs d'attribut enregistrées sur CER	97
3.6 Générateur de code de test CER	16	5.5.3 Le gestionnaire de dépendance demande à CER de recalculer toutes les valeurs d'attributs calculés stockées	98
3.6.1 Méthode de génération du code	18	5.6 Conformité	98
3.7 Outil de couverture de jeu de règles	19	Chapitre 6. A propos de l'éditeur CER	99
3.8 Zone de publication CER	20	6.1 Introduction	99
3.9 Génération de schéma et de catalogue	20	6.2 Menu global de l'éditeur CER	99
3.10 RuleDoc	20	6.2.1 Menu global de l'éditeur CER	99
3.10.1 Exemple simple	21		
3.10.2 Exemple plus pratique	22		
3.10.3 Méthode de génération de RuleDoc	24		
3.11 SessionDoc	24		
3.12 Attributs de règles inutilisés	27		
3.13 Consolidateur de jeu de règles CER	28		
3.13.1 Exemple	28		
Chapitre 4. Gestion des données	31		
4.1 Introduction	31		
4.2 Sessions CER	31		
4.3 Objets de règle externes et internes	33		
4.3.1 Objets de règle externes	34		
4.3.2 Objets de règle internes	37		

6.2.2 Outils de recherche de l'éditeur CER	100
6.2.3 Vue métier	100
6.2.4 Vue technique	101
6.3 Grille du diagramme.	101
6.3.1 Glisser-déposer	101
6.3.2 Basculer sur Afficher les diagrammes détaillés	102
6.3.3 Commandes de panoramique et de zoom	102
6.4 Palettes d'outils et de modèles	102
6.4.1 Palettes applicatives	102
6.4.2 Palette Types de données.	103
6.4.3 Palette technique	104
6.4.4 Modèle d'unités de foyer	106
6.4.5 Modèle d'unités financières	106
6.4.6 Modèle d'unités d'assistance alimentaire	106
6.4.7 Modèle de table de décision.	107
6.5 Menus contextuels d'éléments de règles	107
6.6 Propriétés d'éléments de règles	108
6.6.1 Panneaux de propriété et de validation réductibles	108
6.6.2 Propriétés générales de tous les éléments de règles	108
6.6.3 Propriétés de classe de règles	108
6.6.4 Propriétés d'attribut	109
6.7 Assistants d'éléments de règles	109

Chapitre 7. Référence des éléments de règle pour les palettes de l'éditeur CER

CER	111
7.1 Introduction.	111
7.1.1 Palettes.	111
7.1.2 Palettes réductibles.	111
7.2 Référence des éléments de règle pour les palettes métier étendues et par défaut	111
7.2.1 Rule.	111
7.2.2 And Rule Group	113
7.2.3 Or Rule Group	113
7.2.4 Not	114
7.2.5 Choose	114
7.2.6 Compare	114
7.2.7 When	115
7.2.8 Arithmetic.	115
7.2.9 MIN.	115
7.2.10 MAX	115
7.2.11 SUM	116
7.2.12 Repeating Rule.	116
7.2.13 Filter	116
7.2.14 Size	117
7.2.15 Otherwise	117
7.2.16 Legislation Change	118
7.2.17 Era.	118
7.3 Référence des éléments de règle pour la palette de types de données	118
7.3.1 Boolean	118
7.3.2 String	118
7.3.3 Number	119
7.3.4 Date.	119
7.3.5 Code Table	119
7.3.6 Rate.	120
7.3.7 Frequency Pattern	120

7.3.8 Resource Message	120
7.3.9 Xml Message.	121
7.3.10 Null	121
7.4 Référence des éléments de règle pour la palette de logique technique	122
7.4.1 Create	122
7.4.2 Search	122
7.4.3 Fixed List	123
7.4.4 Property	123
7.4.5 Custom Expression.	124
7.4.6 Existence Timeline	124
7.4.7 Timeline	125
7.4.8 Interval	126
7.4.9 CombineSuccessionSet	126
7.4.10 Call	126
7.4.11 Period Length	127
7.4.12 ALL	128
7.4.13 ANY	128
7.4.14 This	128
7.4.15 Sort	128
7.4.16 Shared Rule Reference	129
7.4.17 CONCAT	130
7.4.18 Join Lists.	130
7.5 Référence des éléments de règle pour les modèles d'unités du foyer	131
7.5.1 Composition du foyer.	131
7.5.2 Catégorie de foyer	131
7.6 Référence des éléments de règle pour les modèles d'unités financières	131
7.6.1 Unité financière	131
7.6.2 Catégorie d'unité financière	131
7.6.3 Membre d'unité financière	131
7.7 Référence des éléments de règle pour les modèles d'unités d'assistance alimentaire	131
7.7.1 Unité d'assistance alimentaire	131
7.7.2 Catégorie d'assistance alimentaire pour une personne seule	132
7.7.3 Catégorie d'assistance alimentaire pour plusieurs personnes	132
7.7.4 Membres de groupe de repas	132
7.7.5 Proches	132
7.7.6 Annuler	132
7.7.7 Membre du foyer	132
7.7.8 Membres facultatifs	132
7.7.9 Exceptions	133
7.8 Référence des éléments de règle pour les modèles de table de décision	133
7.8.1 Decision Table	133

Chapitre 8. CER Best Practices.

8.1 Introduction	135
8.2 Attribut de règle description.	135
8.3 Obtention d'un jeu de règles rapidement fonctionnel	141
8.4 Nommage des éléments de règle.	142
8.5 Moments opportuns à l'utilisation de l'expression reference.	142
8.6 Utilisation de l'outil RuleDoc	143
8.7 Normalisation des règles communes	143
8.8 Suppression des règles inutilisées	144
8.9 Ordre des déclarations	144

8.9.1	Ordre des classes de règles dans un jeu de règles	144
8.9.2	Ordre des attributs de règles calculés dans une classe de règles	144
8.9.3	Ordre des attributs initialisés dans une classe de règles	144
8.9.4	Ordre des conditions booléennes	144
8.10	Création d'objets de règle	145
8.11	Transmission d'objets de règle au lieu d'ID	145
8.12	Développement de méthodes statiques	145
8.13	Eviter les pièges courants lors des tests	150
8.13.1	Eviter assertEquals de JUnit	150
8.13.2	Ne pas oublier d'utiliser .getValue()	151
8.13.3	N'oubliez pas de spécifier toutes les valeurs requises par les calculs testés	152
8.13.4	Ne spécifiez pas la même valeur plusieurs fois	152
8.13.5	Spécification du type correct de valeur pour un attribut	153
8.13.6	Création de tous les objets de règle d'une session avant exécution des calculs getValue	154

Annexe A. Dictionnaire CER XML . . . 157

A.1	Introduction	157
A.2	Jeu de règles	157
A.3	Instruction Include	158
A.4	Classe de règles	159
A.4.1	Attributs initialisés	160
A.4.2	Attributs calculés	160
A.5	Attribut	162
A.6	Expressions	162
A.6.1	Logique booléenne	162
A.6.2	Comparaison de valeurs	162
A.6.3	Constantes	162
A.6.4	Logique conditionnelle	162
A.6.5	Aggrégations de listes	163
A.6.6	Transformations de listes	163
A.6.7	Messages localisables	163
A.6.8	Calculs numériques	163

A.6.9	Références	163
A.6.10	Création	164
A.6.11	Récupération	164
A.6.12	Appels Java	164
A.6.13	Marqueurs	164
A.6.14	Chronologies	164
A.6.15	Eligibilité et autorisation de distribution de produit	164
A.6.16	householdcomposition	164
A.6.17	Liste alphabétique complète des expressions	164
A.7	Annotations	241
A.7.1	Liste alphabétique complète des annotations	241

Annexe B. Opérations de liste utiles 247

Annexe C. Utilisation de CER avec le magasin de données 251

C.1	DataStoreRuleObjectCreator	251
C.2	Exemple	251

Annexe D. Conformité. 257

D.1	API publique de CER	257
D.1.1	Identification de l'API	257
D.1.2	Hors de l'API	257
D.2	Expressions CER	257
D.3	Jeux de règles compris avec l'application	257
D.4	Exemples contenus dans ce guide	258
D.5	Tables de base de données de CER	258
D.5.1	CREOLERuleSet	258
D.5.2	CREOLEMigrationControl	258
D.5.3	Autres tables de base de données de l'infrastructure CER	259
D.6	Gestionnaire de dépendance	259

Remarques 261

Marques	263
-------------------	-----

Figures

1. Sortie d'une chaîne	10	23. Exemple d'utilisation de Timeline et d'Interval pour créer une chronologie	61
2. Sortie d'un message localisable	10	24. Une chronologie des revenus globaux, calculée à l'aide de sum.	63
3. Localisation de "Hello, world!" - anglais	10	25. Exigence liée à une chronologie d'ajout de date	64
4. Localisation de "Hello, world!" - français	11	26. Exigence liée à une chronologie d'extension de dates.	66
5. Interaction avec le jeu de règles localisé	12	27. Exemple de vérification stricte d'une chronologie obtenue.	71
6. Exécution de règles avec l'interpréteur de jeu de règles CER.	14	28. Exemple de vérification plus rapide d'une chronologie obtenue.	73
7. Exécution de règles à l'aide des classes de règles de test générées par CER.	17	29. Commandes de panoramique et de zoom	102
8. Interface Java générée pour la classe de règles HelloWorld.	18	30. Menu contextuel :	131
9. Exemple de rapport de couverture	19	31. Menu contextuel :	132
10. Jeu de règles Hello World.	21	32. SessionDoc indiquant les valeurs description de l'objet de règle	139
11. RuleDoc HelloWorldRuleSet	21	33. SessionDoc pour un objet de règle sans remplacement d'élément description	140
12. RuleDoc de la classe de règles HelloWorld	21	34. Utilisation d'un élément description dans un environnement de développement intégré	141
13. RuleDoc de l'attribut de règle greeting	22	35. Liste sécurisée pour méthodes java.lang.Object	212
14. Calculateur d'année de départ à la retraite	23	36. Safe list for java.lang.Number methods	213
15. RuleDoc indiquant la dérivation et l'utilisation	23	37. Safe list for java.util.List methods	213
16. SessionDoc du test testSelfMadeMillionaireScenario	25	38. Exemple de propriétés - En anglais	223
17. Objets de règles pour le jeu de règles FlexibleRetirementYearRuleSet	25	39. Exemple de propriétés - En français	223
18. SessionDoc de l'objet de règle FlexibleRetirementYear	26	40. Liste sécurisée pour les méthodes java.util.List	247
19. Exemples de données chronologiques.	51	41. Exemple d'entrée provenant d'un fichier CREOLERuleSet.dmx conforme	258
20. Table de vérité pour la règle <i>Chef de famille monoparentale avec mineur à charge</i>	53		
21. Chronologies des situations de Joe, Mary et James	55		
22. Chronologie du statut de Joe, Mary et James en tant que <i>chef de famille monoparentale avec mineur à charge</i>	56		

Tableaux

1.	Description des documents associés	1	37.	Eléments de propriétés de Repeating Rule	116
2.	Calcul des valeurs d'intervalle de la isLoneParentOfMinorTimeline de Mary	75	38.	Eléments des menus contextuels de Repeating Rule	116
3.	Exemple de matrice de dépendance	78	39.	Eléments de propriétés de Filter	117
4.	Exemple de stockage de dépendance	79	40.	Eléments des menus contextuels de Filter	117
5.	Exemple de matrice de dépendance à granularité fine	83	41.	Eléments de propriétés de Size	117
6.	Exemple de matrice de dépendance à granularité grossière	83	42.	Eléments des menus contextuels de l'élément Legislation Change.	118
7.	Eléments précédents identifiés directement par CER	93	43.	Eléments des menus contextuels de Boolean	118
8.	Exemple de dépendances stockées pour les impôts à payer	96	44.	Eléments de propriétés de String	119
9.	Exemple d'éléments de changements précédents pour les impôts à payer	97	45.	Eléments de propriétés de Number	119
10.	Barre de menus	99	46.	Eléments de propriétés de Date	119
11.	Critères de recherches.	100	47.	Eléments de propriétés de Code Table	119
12.	Nouveaux éléments de menu	100	48.	Eléments de propriétés de Rate	120
13.	Nouveaux éléments de menu	101	49.	Eléments de propriétés de Frequency Pattern	120
14.	Eléments de règles sur les palettes applicatives	102	50.	Eléments des menus contextuels de Filter	120
15.	Eléments de règles sur la palette Types de données	104	51.	Eléments de propriétés de Resource Message	121
16.	Eléments de règle sur palette technique	104	52.	Eléments des menus contextuels de Resource Message	121
17.	Eléments de règles sur la palette Modèle d'unités de foyer	106	53.	Eléments de propriétés de XML Message	121
18.	Eléments de règles sur la palette Modèle d'unités financières.	106	54.	Eléments des menus contextuels de Xml Message	121
19.	Eléments de règles sur la palette Modèle d'unités d'assistance alimentaire	106	55.	Eléments de propriétés de Create	122
20.	Eléments sur la palette Table de décision	107	56.	Eléments des menus contextuels de l'élément Create	122
21.	Eléments de menus contextuels généraux	107	57.	Eléments de propriétés de Search	122
22.	Propriétés générales	108	58.	Eléments des menus contextuels de l'élément Search	123
23.	Propriétés de classe de règles	108	59.	Eléments de propriétés FixedList	123
24.	Propriétés d'attribut	109	60.	Eléments de propriétés de Property	124
25.	Table de l'assistant d'élément de règle	110	61.	Eléments des menus contextuels de Property	124
26.	Types de scénarios à utiliser dans les éléments Rule	112	62.	Eléments des menus contextuels de l'élément Custom Expression	124
27.	Eléments de propriétés de Reference	112	63.	Eléments de propriétés de Existence Timeline	125
28.	Eléments des menus contextuels de l'élément Rule	113	64.	Eléments des menus contextuels de l'élément Existence Timeline	125
29.	Eléments des menus contextuels de l'élément And Rule Group	113	65.	Eléments de propriétés d'Existence Timeline	125
30.	Eléments des menus contextuels de l'élément Or	113	66.	Eléments des menus contextuels de l'élément Timeline	125
31.	Eléments des menus contextuels de l'élément Not	114	67.	Eléments des menus contextuels de l'élément Interval	126
32.	Eléments des menus contextuels de l'élément Choose.	114	68.	Eléments des menus contextuels de l'élément CombineSuccessionSet	126
33.	Eléments des menus contextuels de l'élément Compare	114	69.	Eléments de propriétés de Call.	126
34.	Eléments de propriétés d'Arithmetic	115	70.	Eléments des menus contextuels de Call	127
35.	Eléments des menus contextuels de l'élément Min	115	71.	Eléments de propriétés de Period Length	127
36.	Eléments des menus contextuels de l'élément Max.	116	72.	Eléments des menus contextuels de l'élément ALL	128
			73.	Eléments des menus contextuels de l'élément ANY	128
			74.	Eléments des menus contextuels de l'élément ANY	128
			75.	Eléments de propriétés de Shared Rule Reference	129
			76.	Eléments des menus contextuels de l'élément Shared Rule Reference	129

77.	Eléments de propriétés Concat	130	82.	Eléments de propriétés de table de décision	133
78.	Eléments de propriétés Join Lists	130	83.	Eléments des menus contextuels de l'élément	
79.	Eléments des menus contextuels Join Lists	130		Decision Table	133
80.	Eléments des menus contextuels de la				
	catégorie de foyer	131			
81.	Eléments des menus contextuels de la				
	catégorie d'assistance alimentaire pour				
	plusieurs personnes	132			

Avis aux lecteurs canadiens

Le présent document a été traduit en France. Voici les principales différences et particularités dont vous devez tenir compte.

Illustrations

Les illustrations sont fournies à titre d'exemple. Certaines peuvent contenir des données propres à la France.

Terminologie

La terminologie des titres IBM peut différer d'un pays à l'autre. Reportez-vous au tableau ci-dessous, au besoin.

IBM France	IBM Canada
ingénieur commercial	représentant
agence commerciale	succursale
ingénieur technico-commercial	informaticien
inspecteur	technicien du matériel

Claviers

Les lettres sont disposées différemment : le clavier français est de type AZERTY, et le clavier français-canadien de type QWERTY.

OS/2 et Windows - Paramètres canadiens

Au Canada, on utilise :

- les pages de codes 850 (multilingue) et 863 (français-canadien),
- le code pays 002,
- le code clavier CF.

Nomenclature

Les touches présentées dans le tableau d'équivalence suivant sont libellées différemment selon qu'il s'agit du clavier de la France, du clavier du Canada ou du clavier des États-Unis. Reportez-vous à ce tableau pour faire correspondre les touches françaises figurant dans le présent document aux touches de votre clavier.

France	Canada	Etats-Unis
 (Pos1)		Home
Fin	Fin	End
 (PgAr)		PgUp
 (PgAv)		PgDn
Inser	Inser	Ins
Suppr	Suppr	Del
Echap	Echap	Esc
Attn	Intrp	Break
Impr écran	ImpEc	PrtSc
Verr num	Num	Num Lock
Arrêt défil	Défil	Scroll Lock
 (Verr maj)	FixMaj	Caps Lock
AltGr	AltCar	Alt (à droite)

Brevets

Il est possible qu'IBM détienne des brevets ou qu'elle ait déposé des demandes de brevets portant sur certains sujets abordés dans ce document. Le fait qu'IBM vous fournisse le présent document ne signifie pas qu'elle vous accorde un permis d'utilisation de ces brevets. Vous pouvez envoyer, par écrit, vos demandes de renseignements relatives aux permis d'utilisation au directeur général des relations commerciales d'IBM, 3600 Steeles Avenue East, Markham, Ontario, L3R 9Z7.

Assistance téléphonique

Si vous avez besoin d'assistance ou si vous voulez commander du matériel, des logiciels et des publications IBM, contactez IBM direct au 1 800 465-1234.

Chapitre 1. Introduction du manuel de référence

1.1 Objet

Ce manuel de référence décrit le langage des règles Cúram Express Rules (CER), l'environnement de développement et les fonctions d'exécution.

1.2 Public concerné

Ce manuel de référence peut être consulté par toute personne concernée par l'utilisation de CER pour implémenter des règles de calcul applicatives, notamment :

- Les analystes métier, qui rassemblent des exigences impliquant des calculs métier ; la connaissance des possibilités et des approches de CER vous aidera à structurer vos exigences de manière à simplifier l'implémentation dans CER ;
- Les développeurs de jeux de règles, qui sont chargés d'encoder la logique métier d'un jeu de règles ; cette tâche nécessite la compréhension du langage de CER ; et
- Les testeurs, qui sont chargés de vérifier que l'implémentation répond aux exigences ; cette tâche nécessite la compréhension du support de test pour choisir l'approche de vos tests.

Comme c'est souvent le cas, une même personne peut très bien assurer plusieurs de ces rôles (si ce n'est pas tous). En fonction de votre rôle et/ou votre expérience, vous pouvez lire les chapitres du présent manuel dans l'ordre qui vous convient le mieux.

1.3 Documentation connexe

Tableau 1. Description des documents associés

Document	Type de document	Thème associé
Working with Cúram Express Rules	Guide du développeur	Ce guide fournit des instructions détaillées sur la manière de créer des jeux de règles CER dans l'éditeur CER à l'aide d'exemples classés par niveau de complexité des règles.
Inside Cúram Eligibility and Entitlement Using Cúram Express Rules	Guide du développeur	Ce guide explique comment le moteur Cúram Eligibility and Entitlement interagit avec CER pour calculer les déterminations liées aux dossiers de distribution de produit.
How to Build a Product	Guide du développeur	Ce guide décrit toutes les tâches à effectuer dans le cadre de la génération de produit, y compris l'affectation de règles CER à un produit.
Universal Access Customization Guide	Guide du développeur	Ce guide explique comment utiliser les règles CER dans le module Cúram Universal Access pour fournir aux citoyens les résultats d'examen préalable.

1.4 Structure de ce manuel de référence

Ce manuel contient une présentation et des ressources de référence. Il n'est *pas* (forcément) indispensable de le lire dans son intégralité.

Chapitre 2, «Aperçu de CER», à la page 3

Ce chapitre fournit une définition de CER, en explique les avantages et le mode d'utilisation.

Chapitre 3, «Développement et outils de test CER», à la page 9

Ce chapitre décrit les outils disponibles pour le développement et les tests de vos jeux de règles CER.

Chapitre 4, «Gestion des données», à la page 31

Ce chapitre explique comment CER gère et stocke les données, et réagit aux changements de données.

Chapitre 5, «Gestionnaire de dépendance», à la page 77

Ce chapitre décrit comment l'application enregistre qu'une valeur calculée dépend de valeurs d'entrée, et comment ces enregistrements de dépendance sont utilisés pour prendre en charge des recalculs automatiques.

Chapitre 6, «A propos de l'éditeur CER», à la page 99

Ce chapitre décrit les différents composants de l'éditeur CER.

Chapitre 7, «Référence des éléments de règle pour les palettes de l'éditeur CER», à la page 111

Ce chapitre décrit en détails comment créer les éléments d'un jeu de règles dans l'éditeur CER.

Chapitre 8, «CER Best Practices», à la page 135

Ce chapitre fournit des conseils sur la méthode d'écriture de jeux de règles CER *de qualité*.

Annexe A, «Dictionnaire CER XML», à la page 157

Cette annexe constitue une référence liée au format XML utilisé pour le stockage des jeux de règles CER.

Annexe B, «Opérations de liste utiles», à la page 247

Cette annexe décrit quelques opérations utiles figurant sur les listes.

Annexe C, «Utilisation de CER avec le magasin de données», à la page 251

Cette annexe décrit comment CER peut extraire des données du magasin de données.

Annexe D, «Conformité», à la page 257

Cette annexe explique comment assurer le développement avec CER de manière conforme.

Chapitre 2. Aperçu de CER

2.1 Qu'est-ce que CER ?

Définition :

- un langage permettant d'exprimer les règles de calculs métier (dans des jeux de règles) ; et
- un environnement de développement destiné à la création et aux tests de ces jeux de règles.
- un environnement d'exécution pour l'exécution des règles.

2.1.1 Langage des règles

CER est un langage de définition des questions pouvant être posées, et les règles permettant de déterminer les réponses à ces questions.

Chaque question indique :

- son nom ;
- le type de données qui fournit la réponse à la question ; et
- les règles à respecter pour fournir la réponse (si la question est posée).

La réponse à une question peut être aussi simple que oui ou non, par exemple si la question est : Cette personne est-elle éligible au paiement de prestations ? Cependant, CER vous permet de définir le niveau de complexité des types de réponse que vous souhaitez. Par exemple, la question suivante : Quels groupes de personnes du foyer se trouvent en situation d'urgence ? aurait pour réponse une liste de groupes du foyer, chacun contenant une liste de personnes.

Les règles permettant de déterminer la réponse à une question peuvent être aussi simples ou aussi complexes que vous le souhaitez. Par exemple, la règle de la réponse à la question : Quelle est la date de naissance du demandeur ? serait probablement (en des termes simples) : La date que le demandeur a déclarée comme étant sa date de naissance. En revanche, la règle de la réponse à la question : Cette personne est-elle éligible au paiement de prestations ? est susceptible d'impliquer d'autres questions telles que : Quel est le niveau de revenus de cette personne ? et/ou Combien d'enfants cette personne a-t-elle ?.

CER applique sa propre terminologie à ces concepts :

- **Classe de règles**

Une classe de règles est un type d'objet qui contient des données, tel qu'une personne, des revenus ou une réclamation. Une nouvelle classe de règles peut être créée dans l'éditeur CER. Voir 6.2.4, «Vue technique», à la page 101

- **Objet de règle**

Un objet de règle est une instance d'une classe de règles, par exemple John Smith (personne), les revenus de John Smith issus de son emploi à temps partiel (revenus), ou une demande de John Smith pour des prestations d'aide pour enfants (réclamation).

- **Attribut de règle**

Un attribut de règle est une question pouvant être posée. Il est défini sur une classe de règles et peut être demandé à tout objet de règle de cette classe. Par exemple, la classe de règles Personne peut définir l'attribut de règle dateOfBirth, et l'on peut donc demander à l'objet de règle John Smith son dateOfBirth (par exemple, le 3 octobre 1970). Un nouvel attribut peut être créé pour la classe de règles sélectionnée dans l'éditeur CER. Voir 6.2.4, «Vue technique», à la page 101.

- **Expression**

Une expression est une étape de calcul qui peut être utilisée pour répondre à une question. Par exemple, si l'éligibilité d'une réclamation dépend du fait que les revenus totaux d'une personne se

situent en dessous d'un certain seuil, il est possible d'utiliser une expression Sum pour calculer les revenus totaux, puis une expression Compare pour comparer ce total avec le montant du seuil. Pour créer une expression, vous pouvez déplacer un élément de règle Sum vers l'attribut de règle dans l'éditeur CER. Voir 6.2.3, «Vue métier», à la page 100

- **Jeu de règles**

Un jeu de règles est un ensemble ¹de classes de règles, généralement centrées sur des objectifs spécifiques. Par exemple, un jeu de règles lié à une détermination de prestations pour enfants peut inclure les classes de règles Réclamation, Personne et Revenus. Un nouveau jeu de règles peut être créé dans la section Règles et informations collectées de l'interface d'administration.

- **Session de règles**

Une session de règles contrôle l'exécution de règles. Par exemple, votre application peut créer une session de règles afin de déterminer l'éligibilité de John Smith aux prestations pour enfants, en appelant le jeu de règles approprié et en posant des questions d'éligibilité concernant la situation personnelle de John.

2.1.2 Création et environnement de test

Les jeux de règles CER sont créés et gérés dans l'éditeur CER. Ils sont stockés sous forme de données XML sur la base de données de l'application. Les données XML d'un jeu de règles CER sont conformes au schéma de règles fourni par CER.

CER inclut également un valideur de jeu de règles global qui peut détecter des erreurs dans votre jeu de règles avant d'autoriser l'exécution de vos règles. Vous pouvez valider votre jeu de règles dans l'éditeur CER. Voir 6.2, «Menu global de l'éditeur CER», à la page 99.

CER prend en charge l'exécution de sessions de règles dans :

- les environnements de production, dans lesquels CER s'intègre à votre application pour répondre aux questions lorsque cela est nécessaire ; et
- un environnement de test autonome sur lequel vous créez des tests réitérables automatisés pour vos jeux de règles.

Les jeux de règles CER sont intégralement dynamiques. Dans les environnements de production, CER prend en charge le téléchargement des changements apportés aux jeux de règles qui prennent effet lorsqu'ils sont publiés ; aucune reconstitution ou le redéploiement de votre application n'est requis.

Le test des jeux de règles CER peut être effectué au niveau qui vous convient. Vous pouvez choisir de fournir des données de test détaillées pour un scénario métier complet et/ou vous pouvez créer des tests isolés pour des composants de votre jeu de règles *sans* devoir configurer soigneusement de grandes quantités de données d'entrée.

Par exemple, la détermination de l'éligibilité d'une personne à des prestations pour enfants peut représenter un calcul complexe impliquant notamment la comparaison des revenus totaux de la personne à un certain seuil. En outre, le calcul des revenus totaux de la personne est *lui-même* un calcul complexe, qui implique des décisions déterminant si certains types de revenus peuvent être pris en compte pour déterminer l'éligibilité à la prise en charge de l'enfant.

Lorsque vous testez le calcul d'éligibilité, dans le cadre d'un développement traditionnel, vous pouvez être amené à configurer soigneusement les données de revenus pour obtenir un total de revenus calculé, que vous pouvez alors utiliser pour tester le calcul d'éligibilité. En fonction de la complexité des calculs, cette configuration de données peuvent être très fastidieuse et très délicate à modifier.

1. Depuis Cúram V6, les jeux de règles ne sont plus autonomes. Une classe d'un jeu de règles peut étendre une classe de règles issue d'un autre jeu de règles ; le type de données d'un attribut de règle dans un jeu de règles peut être une classe de règles issue d'un autre jeu de règles ; et les expressions permettant de lire ou de créer des objets de règle peuvent utiliser des classes de règles issues d'autres jeux de règles.

Par comparaison, dans CER vous pouvez simplement remplacer un calcul sans avoir à fournir des données détaillées de bas niveau ; dans CER, il est extrêmement facile de générer un test qui signifie effectivement : pour les besoins de ce test, les revenus totaux sont de 10 \$ - ne tentez *pas* de calculer les revenus totaux au cours de ce test.

Cette fonction de CER permet donc de tester facilement toutes les fonctionnalités de votre jeu de règles à un niveau pertinent.

L'environnement de création comprend également des outils pour vous aider dans le développement et les tests de vos règles :

- **RuleDoc**
Extraction HTML de la structure de vos jeux de règles.
- **SessionDoc**
Représentation HTML des données de vos objets de règle.
- **Outil de couverture**
Indique quelle étendue de votre jeu de règles a fait l'objet de vos tests.

2.1.3 Environnement d'exécution

CER exécute des règles sur demande au moment de l'exécution.

Depuis Cúram V6, CER possède également des fonctions permettant :

- de stocker les objets de règle dans la base de données, de sorte que les objets de règle soient disponibles pour traitement futur ; et
- de s'intégrer au gestionnaire de dépendance pour détecter la date de changement des données d'entrée, et de mettre à jour automatiquement les résultats des calculs qui dépendent de ces éléments de données d'entrée (à l'instar du traitement de feuille de calcul traditionnel).

2.2 Quelles sont ses avantages ?

CER offre des avantages clés :

- **Simplicité**
Les jeux de règles CER ne sont pas plus complexes que vos besoins métier. Les utilisateurs métier et les utilisateurs techniques peuvent lire les jeux de règles CER et en comprendre aisément la signification. Les jeux de règles sont simples à écrire et simples à tester.
- **Flexibilité**
Les jeux de règles sont faciles à changer. Vous pouvez ajouter de nouvelles questions à tout moment et CER garantit l'absence d'impact sur le comportement existant. Vous pouvez apporter des changements au mode de réponse à une question existante, et CER vous montre les calculs qui dépendent de cette question. Ainsi, vous contrôlez parfaitement l'effet de vos changements.
- **Support de localisation**
CER peut produire une sortie localisable, de sorte que les réponses aux questions peuvent être affichées à vos utilisateurs finaux en fonction de leurs préférences de langue et d'environnement local.
- **Validité**
CER assure la détection d'erreurs dans votre jeu de règles avant de l'exécuter. Le valideur de jeu de règles CER signale autant d'erreurs que possible pour que vous puissiez les corriger en une seule fois. CER recherche les problèmes techniques de votre jeu de règles de sorte que vous puissiez vous concentrer uniquement sur les fonctionnalités de votre jeu de règles.
- **Testabilité**
Vous pouvez tester vos jeux de règles CER à la granularité qui vous convient. CER vous permet de garder le contrôle de vos jeux de règles volumineux par la création de tests pour les sections discrètes de vos règles.

- **Support dynamique**

Vous pouvez apporter des changements à votre jeu de règles CER dans un système en cours d'exécution et vos changements prennent effet immédiatement au moment de la publication.

- **Comportement similaire aux feuilles de calcul**

La construction de règles CER est comparable à la disposition des formules dans les cellules de feuille de calcul (que de nombreux utilisateurs connaissent). En cas de changement des données d'entrée (telles que les informations collectées, les données personnelles ou les taux de paiement), CER s'intègre au gestionnaire de dépendance pour recalculer automatiquement les valeurs dérivées qui sont concernées par le changement.

2.3 Mode d'utilisation

L'environnement de développement CER s'intègre étroitement à un environnement de développement d'applications plus large.

CER est utilisé par un certain nombre de domaines d'application, y compris (mais non limité à) :

- **Module Cúram Universal Access**

Le module Cúram Universal Access repose sur CER pour déterminer l'éligibilité potentielle aux programmes des services sociaux lorsque les citoyens utilisent le module en libre-service pour effectuer un auto-examen préalable. En fonction des données capturées, CER détermine à quels programmes le citoyen est potentiellement éligible, et fournit un texte (dans la langue du citoyen) expliquant *pourquoi* le citoyen est ou non potentiellement éligible.

- **Assistant**

L'assistant utilise des règles CER pour calculer les conseils à afficher aux utilisateurs. Ces conseils sont automatiquement mis à jour lorsque la situation du citoyen change.

- **Moteur Eligibility and Entitlement**

Le moteur Case Eligibility and Entitlement de Cúram s'intègre étroitement à CER pour déterminer l'éligibilité et les droits liés à un dossier (et expliquer les résultats obtenus) sur toute la durée de vie du dossier. Le moteur Case Eligibility and Entitlement repose sur l'intégration entre CER et le gestionnaire de dépendance pour identifier à quel moment recalculer la détermination d'un dossier, soit en raison de changements spécifiques au dossier, telles que les données d'informations collectées, ou de changements de données plus générales, telles que ceux apportés aux données personnelles ou aux données de taux de produit.

Vous pouvez également utiliser CER pour effectuer des calculs liés à vos propres domaines métier personnalisés.

L'environnement d'exécution CER ne comprend pas de concepts métier intégrés ; en revanche, chaque domaine métier communique avec CER via l'utilisation des éléments suivants :

- des classes de règles d'interface qui encapsulent les concepts métier ; et/ou
- des extensions au langage de CER qui reconnaissent les concepts métier.

Pour plus de détails sur la façon dont les composants d'application utilisent CER, voir les guides commerciaux et techniques de ce composant.

2.4 Principes de base

CER repose sur certains principes clés. Une certaine connaissance de ces principes peut vous aider à comprendre l'approche CER :

- **Les questions obtiennent une réponse uniquement lorsque nécessaire**

Le travail effectué pour répondre à une question est effectué uniquement lorsque cette question est posée.

- **Toutes les données sont non modifiables**

La réponse à une question est une valeur qui ne peut pas être changée accidentellement par un élément externe à CER. Si la réponse à une question est recalculée, une nouvelle valeur de réponse est créée.

- **Autoriser plusieurs questions**

Un jeu de règles ne s'exécute pas en une seule fois, mais il autorise autant de questions que nécessaire.

- **Spécifier les règles, pas l'ordre d'exécution**

Vous pouvez spécifier les règles permettant de répondre à une question, et laisser CER répondre efficacement à ces questions lors de l'exécution.

- **Pas de volatilité**

Les entrées identiques traitées par des règles identiques produisent toujours le même résultat.

- **Pas de mémoire de travail**

Il n'existe aucun compteur ou totaux cumulatifs. Un nombre ou un total constitue une question à part entière : vous fournissez les règles permettant d'y répondre, et CER détermine l'ordre d'exécution lors de la réponse.

- **Nommer ses besoins**

Il vous suffit de donner des noms aux concepts métier et aux questions. Il n'est pas nécessaire de trouver des noms descriptifs pour les résultats intermédiaires (sauf si vous le souhaitez).

- **Le développement et les tests vont de pair**

CER offre un support solide de gestion des tests de vos règles.

- **Pas de concepts métier intégrés**

L'environnement d'exécution CER est conçu sans aucun concept métier. Vous pouvez définir les concepts métier dont vous avez besoin, ce qui fait de CER un environnement d'exécution d'usage courant.

- **L'implémentation des règles correspond parfaitement aux exigences qui y sont liées**

L'implémentation de vos exigences de règle est aussi complexe que ces exigences : *ce n'est plus le cas*. Les jeux de règles CER sont compréhensibles lorsqu'elles sont visualisées par les analystes métier qui ont rassemblé les exigences initiales.

- **Se baser sur le support Java familier**

CER ne réinvente pas la roue : les fonctionnalités proposées par la technologie Java™ existante sont facilement réutilisées dans les jeux de règles CER.

- **Gestion des dépendances de calcul**

CER s'intègre au gestionnaire de dépendance pour gérer les dépendances de calcul, de sorte que vous n'avez pas à le faire. Lors d'un changement d'élément de données d'entrée, le gestionnaire de dépendance et CER connaissent les éléments à recalculer. Il n'est pas nécessaire d'écrire de traitement particulier visant à évaluer quelles sorties calculées *peuvent* avoir été affectées.

Chapitre 3. Développement et outils de test CER

3.1 Introduction

Ce chapitre décrit les outils disponibles pour le développement et le test de vos jeux de règles CER.

3.2 Editeur de règles CER

L'éditeur CER fournit un environnement et une interface conviviaux permettant aux techniciens et aux professionnels de créer, éditer et valider un jeu de règles et ses classes de règles. Voir Chapitre 6, «A propos de l'éditeur CER», à la page 99 pour en savoir plus sur l'utilisation de l'éditeur CER.

3.3 Prise en charge de la localisation

La localisation dans CER couvre ces deux tâches distinctes :

- localisation des données calculées renvoyées par un attribut de règle CER, de manière à ce que la sortie puisse s'afficher pour les utilisateurs dans différents environnements locaux et
- localisation des descriptions concernant les artefacts dans vos jeux de règles CER, afin que les utilisateurs affichant les jeux de règles dans l'éditeur CER puissent faire de même dans leur propre environnement local.

Ces éléments sont traités de manière plus détaillée dans les sections suivantes.

Important : Les *noms* des éléments de règles tels que les classes et les attributs de règles *ne peuvent pas* être localisés, étant donné qu'ils sont utilisés au sein du langage CER en tant qu'identificateurs, tels que le nom d'un attribut dans une expression référence.

Par contre, les *descriptions* des éléments de règle peuvent être localisées, en laissant les noms des éléments inchangés.

3.3.1 Localisation des données calculées

CER prend en charge la classe Java ordinaire `String`.

Les chaînes peuvent être utiles dans les phases initiales de développement de votre jeu de règles ; cependant, si vos règles contiennent une sortie qui doit s'afficher pour les utilisateurs dans différents environnements locaux, vous pouvez avoir besoin de recourir à la prise en charge de la localisation proposée par CER.

La valeur de chaîne «Hello World» correspondant à l'exemple ci-dessous convient aux utilisateurs finaux qui comprennent l'anglais ; mais qu'en est-il pour ceux qui ne le comprennent pas ?

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

Figure 1. Sortie d'une chaîne

CER inclut une interface `curam.creole.value.Message` qui permet la conversion d'une valeur vers une chaîne spécifique à un environnement local lors de l'exécution.

Pour obtenir une liste des expressions CER capables de créer une instance de `curam.creole.value.Message`, voir A.6.7, «Messages localisables», à la page 163.

Maintenant, nous allons réécrire l'exemple `HelloWorld` afin de le rendre localisable :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="LocalizableHelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <!-- Utiliser un message, non une chaîne -->
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Rechercher une valeur à partir d'une propriété
          localisable, au lieu de coder en dur une
          chaîne en langue unique -->
        <ResourceMessage key="greeting"
          resourceBundle="curam.creole.example.HelloWorld"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

Figure 2. Sortie d'un message localisable

```

# file curam/creole/example/HelloWorld_en.properties
greeting=Hello, world!

```

Figure 3. Localisation de "Hello, world!" - anglais

```
# file curam/creole/example/HelloWorld_fr.properties  
greeting=Bonjour, monde!
```

Figure 4. Localisation de "Hello, world!" - français

Comment ce message va-t-il se comporter lors de l'exécution ? Un code qui interagit avec votre jeu de règles doit appeler la méthode `toLocale` sur tous types de messages, pour les convertir dans l'environnement local requis.

```

package curam.creole.example;

import java.util.Locale;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
    curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld;
import
    curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Message;

public class TestLocalizableHelloWorld extends TestCase {

    /**
     * Exécute la classe en tant qu'application Java autonome.
     */
    public static void main(final String[] args) {

        final TestLocalizableHelloWorld testLocalizableHelloWorld =
            new TestLocalizableHelloWorld();
        testLocalizableHelloWorld.testLocalizedRuleOutput();

    }

    /**
     * Scénario de test simple, affichant la sortie localisée dans différents environnements
     * locaux.
     */
    public void testLocalizedRuleOutput() {

        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        final HelloWorld helloWorld =
            HelloWorld_Factory.getFactory().newInstance(session);

        // renvoie un message, non une chaîne
        final Message greeting = helloWorld.greeting().getValue();

        // pour décoder le message, nous devons utiliser l'environnement local de l'utilisateur
        final String greetingEnglish =
            greeting.toLocale(Locale.ENGLISH);
        final String greetingFrench = greeting.toLocale(Locale.FRENCH);

        System.out.println(greetingEnglish);
        System.out.println(greetingFrench);

        assertEquals("Hello, world!", greetingEnglish);
        assertEquals("Bonjour, monde!", greetingFrench);
    }
}

```

Figure 5. Interaction avec le jeu de règles localisé

S'ils sont utilisés dans un message localisable, les types de données suivants sont mis en forme lors de l'exécution en tenant compte de l'environnement local :

- Objets de règles (utilisant la valeur de l'attribut description de l'objet de règle) ;
- Dates (utilisant `curam.util.type.Date`) ;
- Éléments de table de code et
- Messages localisables imbriqués.

Tous les autres objets s'affichent à l'aide de leur méthode `toString`.

3.3.2 Localisation des descriptions d'artefacts de règles CER

L'éditeur CER vous permet de placer une description sur ces artefacts de jeu de règles (via l'annotation "Label") :

- Jeu de règles ;
- Classe de règles ;
- Attribut de règle et
- Expression.

Lorsqu'un jeu de règles CER est publié à l'aide de l'application d'administration Cúram, les descriptions sur ces artefacts de jeu de règles sont stockées dans le magasin de ressources de l'application en tant que fichiers de propriétés (nommé RULESET- (nom du jeu de règles) (numéro de version du jeu de règles)). Vous pouvez localiser ces fichiers de propriétés de la même façon que pour toute autre ressource dans le magasin de ressources.

La prise en charge de la localisation des artefacts de règles CER via l'éditeur CER sera incluse dans une édition ultérieure de l'éditeur CER.

3.4 Valideur de jeu de règles

CER contient un valideur qui vérifie la structure de vos jeux de règles. Généralement, la validation de vos jeux de règles s'effectue à l'aide de l'application d'administration Cúram et/ou de l'éditeur CER.

Concernant les jeux de règles sur le système de fichiers, vous pouvez exécuter cette commande pour valider la structure de ces jeux de règles :

Exécutez la commande suivante :

```
build creole.validate.rulesets
```

La cible exécute alors le valideur de jeu de règles CER sur vos jeux de règles pour signaler les erreurs et/ou avertissements éventuels concernant vos jeux de règles CER.

Conseil : Le valideur de jeu de règles CER signale également les avertissements concernant les problèmes non critiques dans vos jeux de règles. Ces avertissements ne vous empêchent pas d'exécuter et de tester vos règles, mais doivent être traités pour garantir un jeu de règles optimal.

3.5 Interpréteur de jeu de règles

CER contient un interpréteur capable d'exécuter des jeux de règles définis dynamiquement.

Vous trouverez ci-dessous un exemple de code qui utilise l'interpréteur de jeu de règles CER pour exécuter des règles à partir d'un jeu de règles `HelloWorldRuleSet`.

```

package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.RuleObject;
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import curam.creole.parser.RuleSetXmlReader;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldInterpreted extends TestCase {

    /**
     * Exécute la classe en tant qu'application Java autonome.
     */
    public static void main(final String[] args) {

        final TestHelloWorldInterpreted testHelloWorld =
            new TestHelloWorldInterpreted();
        testHelloWorld.testUsingInterpreter();
    }

    /**
     * Lit le jeu de règles HelloWorldRuleSet à partir de son fichier source XML.
     */
    private RuleSet getRuleSet() {

        /* Chemin d'accès relatif au fichier source du jeu de règles */
        final String ruleSetRelativePath = "./rules/HelloWorld.xml";

        /* lecture de la source du jeu de règles */
        final RuleSetXmlReader ruleSetXmlReader =
            new RuleSetXmlReader(ruleSetRelativePath);

        /* exportation des problèmes éventuels */
        ruleSetXmlReader.validationProblemCollection().printProblems(
            System.err);

        /* échec en cas d'erreurs dans le jeu de règles */
        assertTrue(!ruleSetXmlReader.validationProblemCollection()
            .containsErrors());

        /* renvoie le jeu de règles à partir du lecteur */
        return ruleSetXmlReader.ruleSet();
    }

    /**
     * Scénario de test simple, utilisant l'interpréteur de jeu de règles CER
     * dynamique complet.
     */
    public void testUsingInterpreter() {

        /* lecture du jeu de règles */
        final RuleSet ruleSet = getRuleSet();

        /* démarre une session qui crée des objets de règles interprétés */
        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new InterpretedRuleObjectFactory()));

        /* crée une instance d'objet de règle de la classe de règles obligatoire */
        final RuleObject helloWorld =
            session.createRuleObject(ruleSet.findClass("HelloWorld"));

        /*
         * Accès à l'attribut de règle "greeting" sur l'objet de règle -
         * le résultat doit être transtypé selon le type attendu (String)
         */
        final String greeting =

```

Vous pouvez exécuter cet exemple de classe en tant qu'application Java autonome (c'est-à-dire via sa méthode main) ou en tant que test JUnit. Ces deux manières d'exécuter cette classe sont proposées simplement pour plus de praticité ; aussi, lorsque vous écrivez votre propre code pour exécuter des jeux de règles, choisissez celle qui vous convient le mieux.

Examinons maintenant ce code en détail. La méthode de test `testUsingInterpreter` effectue les fonctions clés suivantes :

- elle lit le jeu de règles ;
- elle démarre une session CER ;
- elle crée une nouvelle instance d'objet de règle dans la session et
- elle exécute des règles en récupérant une valeur d'attribut à partir de l'objet de règle.

3.5.1 Lecture du jeu de règles

```
/* lecture du jeu de règles */
    final RuleItem_RuleSet ruleSet = getRuleSet();
```

Cette ligne appelle une méthode utilitaire pour lire le jeu de règles à partir d'un fichier source XML.

Ce jeu de règles est explicitement validé pour garantir qu'il ne contient pas d'erreur :

```
/* exportation des problèmes éventuels */
    ruleSetXmlReader.validationProblemCollection().printProblems(
        System.err);

    /* échec en cas d'erreurs dans le jeu de règles */
    assertTrue(!ruleSetXmlReader.validationProblemCollection()
        .containsErrors());
```

3.5.2 Démarrage d'une session CER

```
/* démarre une session qui crée des objets de règles interprétés */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new InterpretedRuleObjectFactory()));
```

Ces lignes créent une nouvelle session CER pour le jeu de règles.

Une session gère les objets de règles créés pour les classes dans le jeu de règles. Dans cet exemple, nous utilisons une session qui crée des objets de règles dynamiques complets (à l'aide de l'élément `InterpretedRuleObjectFactory`). Comme on le voit ci-dessous, dans une session interprétée, chaque référence à une classe de règles ou à un nom d'attribut s'effectue via un appel API prenant des noms semblables à un paramètre String.

3.5.3 Créer un nouvel objet de règle

```
/* crée une instance d'objet de règle de la classe de règles obligatoire */
    final RuleObject helloWorld =
        session.createRuleObject("HelloWorld");
```

Cette ligne crée un nouvel objet de règle (une instance de la classe de règles "HelloWorld") et stocke l'objet de règle dans la mémoire de la session CER.

3.5.4 Exécution de règles

```
/*
 * Accès à l'attribut de règle "greeting" sur l'objet de règle -
 * le résultat doit être transtypé selon le type attendu (String)
 */
final String greeting =
(String) helloWorld.getAttributeValue("greeting")
.getValue();
```

Cette ligne récupère la valeur de l'attribut "greeting" à partir de l'objet de règle créé ci-dessus.

Lorsque la valeur de l'attribut est demandée, CER exécute les règles pour générer la valeur de l'attribut (dans ce cas, en renvoyant la chaîne constante "Hello, world!").

Remarque : Lors de l'exécution d'une session interprétée, vous pouvez transtyper la sortie de `getValue` vers le type de données attendu.

Dans cet exemple, nous avons demandé la valeur d'un attribut seulement ; cependant, même si la session est toujours active, le code peut demander la valeur de n'importe quel attribut sur n'importe quel objet de règle dans la session. CER retient les valeurs déjà calculées et effectue un calcul seulement la première fois que celui-ci lui est demandé.

3.6 Générateur de code de test CER

L'interpréteur de jeu de règles CER ci-dessus autorise la référence aux noms de classe de règles et d'attribut via des chaînes.

Même si celui-ci permet une configuration dynamique complète des jeux de règles, dans le cadre du test, il peut s'avérer fastidieux d'avoir à utiliser des chaînes et des valeurs d'attribut de transtypage. Si vous faites une faute de frappe dans le nom d'une classe de règles ou d'un attribut, ou si utilisez le mauvais type de transtypage, votre code peut être compilé sans erreur mais vous rencontrerez des erreurs lors de l'exécution.

CER inclut un générateur de code capable de générer des classes d'encapsuleur Java pour vos classes de règles. Ces classes générées peuvent simplifier l'écriture de votre code de test et permettre au compilateur de détecter les problèmes qui se produiraient uniquement lors de l'exécution.

Nous allons à présent réécrire notre code pour exécuter le jeu de règles `HelloWorldRuleSet` :

```

package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.ruleclass>HelloWorldRuleSet.impl>HelloWorld;
import
    curam.creole.ruleclass>HelloWorldRuleSet.impl>HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldCodeGen extends TestCase {

    /**
     * Exécute la classe en tant qu'application Java autonome.
     */
    public static void main(final String[] args) {

        final TestHelloWorldCodeGen testHelloWorld =
            new TestHelloWorldCodeGen();
        testHelloWorld.testUsingGeneratedTestClasses();

    }

    /**
     * Scénario de test simple, utilisant les classes de test générées par CER pour
     * les tests de typage fort et de simplicité de codification.
     */
    public void testUsingGeneratedTestClasses() {

        /* démarre une session à typage fort */
        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        /*
         * crée une instance d'objet de règle de la classe de règles obligatoire, à
         * l'aide de sa classe générée
         */
        final HelloWorld helloWorld =
            HelloWorld_Factory.getFactory().newInstance(session);

        /*
         * utilise l'accessor généré pour obtenir l'attribut de règle
         * "greeting" (aucun transtypage nécessaire) et toute erreur dans le
         * nom d'attribut provoque une erreur de compilation
         */
        final String greeting = helloWorld.greeting().getValue();

        System.out.println(greeting);
        assertEquals("Hello, world!", greeting);
    }

}

```

Figure 7. Exécution de règles à l'aide des classes de règles de test générées par CER

Remarquez les comparaisons suivantes avec notre code TestHelloWorldInterpreted :

- la session employée avec le générateur de code utilise StronglyTypedRuleObjectFactory, nommé ainsi car il traite les classes Java générées plutôt que les instances RuleObject dynamiques ;
- le chargement du jeu de règles n'est pas obligatoire (d'où l'absence de méthode utilitaire) ;

- la référence à la classe de règles HelloWorld s'effectue via une interface Java du même nom ; toute faute de frappe dans le nom sera détectée par le compilateur Java ;
- de la même manière, la référence à l'attribut de règle greeting s'effectue via une méthode Java du même nom sur l'interface ;
- aucun transtypage du type de retour n'est requis, car la méthode greeting retourne le type correct (String).

avertissement : Le code généré est uniquement destiné à être utilisé dans les environnements de test où il est simple de recompiler les changements apportés au code.

Le code généré n'est *pas* transférable entre les machines, car il contient des chemins d'accès absolus aux jeux de règles sur la machine locale.

En particulier, vous ne *devez pas* utiliser le code généré dans un environnement de production où les jeux de règles sont susceptibles de changer dynamiquement.

3.6.1 Méthode de génération du code

Pour exécuter le générateur de code CER, exécutez la commande suivante :

```
build creole.generate.test.classes
```

La cible exécutera également le valideur de jeu de règles CER sur vos jeux de règles. En cas d'erreurs, le valideur de jeu de règles CER signale les erreurs et les arrêts de traitement. S'il n'y a pas d'erreur, le générateur CER sort des classes Java générées et des interfaces pour vos jeux de règles et classes de règles CER.

Conseil : Le valideur de jeu de règles CER signale également les avertissements concernant les problèmes non critiques dans vos jeux de règles. Ces avertissements ne vous empêchent pas d'exécuter et de tester vos règles, mais doivent être traités pour garantir un jeu de règle optimal.

Le générateur de code CER place sa sortie dans le répertoire EJBServer/build/svr/creole.gen/source.

Voici un exemple d'interface générée pour la classe de règles HelloWorld :

```
/*
 * Générée par le générateur de code Curam CREOLE
 * Générateur Copyright 2008-2010 Curam Software Ltd.
 */
package curam.creole.ruleclass.HelloWorldRuleSet.impl;
/**
 * Interface générée par le code pour les tests.
 * <p/>
 * Les clients ne doivent pas implémenter cette interface.
 */
public interface HelloWorld extends
curam.creole.execution.RuleObject {
/**
 * Accesseur généré par le code pour les tests.
 * @renvoie le conteneur de la valeur d'attribut greeting
 */
public curam.creole.execution.AttributeValue<? extends
java.lang.String> greeting();
}
```

Figure 8. Interface Java générée pour la classe de règles HelloWorld

Conseil : Vous devez régénérer vos classes de test si vous apportez des changements structurels à vos jeux de règles sur le système de fichiers, par exemple :

- création d'un nouveau jeu de règles ou suppression d'un jeu de règles existant ;
- ajout d'une nouvelle classe de règles à un jeu de règles ou suppression d'une classe de règles existante d'un jeu de règles ;
- ajout d'un nouvel attribut de règle à une classe de règles ou suppression d'un attribut de règle existant d'une classe de règles ;
- changement de la valeur "extends" pour une classe de règles existante et/ou
- changement du type de données d'un attribut.

Vous n'avez *pas* besoin de régénérer les classes de test si vos changements sont limités à l'*implémentation* d'un attribut de règle (c'est-à-dire à ses expressions de dérivation). Les dérivations sont toujours traitées dynamiquement à partir du jeu de règles lors de l'exécution et ne sont pas présentes dans les classes de test générées.

3.7 Outil de couverture de jeu de règles

CER inclut un outil permettant de rapporter les composants d'un jeu de règles qui sont "couverts" lors de l'exécution. Les statistiques de couverture peuvent être rapportées pour n'importe quel traitement nécessitant des valeurs de la part de CER, notamment :

- l'application en ligne en cours d'exécution et
- les exécutions de tests JUnit.

Pour capturer les données de couverture, définissez la propriété d'environnement `curam.creole.coverage.logfile` (dans `Bootstrap.properties`) sur l'emplacement d'un fichier. Tandis que les règles s'exécutent, les lignes contenant des informations de couverture sont ajoutées au fichier lors de l'évaluation des expressions CER.

Conseil : Pour effacer les données de couverture, il vous suffit de supprimer le fichier indiqué dans votre paramètre `curam.creole.coverage.logfile`.

Au fil du temps, le fichier de données de couverture peut devenir volumineux, c'est pourquoi vous devez désactiver la capture des données de couverture lorsqu'elle n'est pas obligatoire en supprimant (ou en mettant en commentaire) votre paramètre `curam.creole.coverage.logfile`.

Pour créer un rapport de couverture, exécutez la cible suivante :

```
build creole.report.coverage -Dfile.coverage.log= emplacement de fichier
```

Un rapport détaillé codé en couleur simple est écrit dans `.../EJBServer/build/svr/creole.gen/coverage/index.html`, où

- Vert = couvert
- Jaune = partiellement couvert
- Rouge = non couvert

Les attributs de règles ayant une dérivation de type `<specified>` sont volontairement exclus du rapport. Un exemple de rapport est proposé ci-dessous :

Figure 9. Exemple de rapport de couverture

Remarque : Les jeux et classes de règles qui sont "inclus" dans d'autres jeux de règles (à l'aide du mécanisme `<Include>`) deviennent essentiellement des composants de la source d'éléments extérieurs, notamment le jeu de règles. Il convient de garder cela à l'esprit lors de l'analyse des rapports de couverture.

3.8 Zone de publication CER

L'application d'administration Cúram contient des écrans permettant de répertorier les ensembles de règles CER. A partir de cette application, vous pouvez :

- afficher un jeu de règles existant (dans l'éditeur CER), puis consulter les versions historiques d'un jeu de règles CER donné ;
- une fois qu'il est ouvert, apportez des changements à un jeu de règles existant (dans l'éditeur CER) ;
- créez un nouveau jeu de règles (et ouvrez-le dans l'éditeur CER pour ajouter des règles) et/ou
- supprimez un jeu de règles existant.

A compter de Cúram V6, les changements apportés aux jeux de règles CER ne s'appliquent *pas* immédiatement, mais sont stockés dans la zone de publication jusqu'à ce qu'ils soient publiés.

Vous pouvez accumuler plusieurs jeux de règles CER dans cette zone ; en effet, vous pouvez *devoir* accumuler des changements liés à plusieurs jeux de règles, si le changement que vous effectuez affecte plusieurs jeux de règles.

Vous pouvez décider de valider vos changements en attente à tout moment. Lorsque les changements vous conviennent, vous pouvez décider de les publier. Le système confirme la validité des jeux de règles, puis autorise la poursuite de la publication.

La publication des changements du jeu de règles CER se produit lors d'un traitement différé, étant donné que les objets de règles CER existants devront être mis à jour en fonction des changements apportés aux classes de règles CER et/ou des recalculs mis en file d'attente pour des attributs dont les dérivations ont changé.

3.9 Génération de schéma et de catalogue

Le schéma des jeux de règles CER est assemblé dynamiquement pour prendre en compte :

- le schéma fixe des jeux de règles, classes et attributs CER et
- les contributions aux expressions et annotations CER par les composants d'application.

En général, le schéma assemblé dynamiquement réside dans la mémoire lorsque CER effectue le traitement de validation. Cependant, dans certains cas, il peut être utile de disposer de ce schéma (et d'un catalogue l'indiquant) sur le système de fichiers.

Pour générer le fichier de schéma CER (EJBServer/build/svr/creole.gen/schema/RuleSet.xsd), exécutez la commande suivante :

```
build creole.generate.schema
```

Pour générer un catalogue (EJBServer/build/svr/creole.gen/catalog/CREOLECatalog.xml) indiquant le fichier de schéma CER, exécutez la commande suivante :

```
build creole.generate.catalog
```

3.10 RuleDoc

CER inclut un outil permettant de générer automatiquement la documentation à partir de vos jeux de règles et classes de règles CER. Cette documentation est connue sous le nom de RuleDoc.

RuleDoc peut être particulièrement utile dans les cas suivants :

- discussion au sujet du comportement de votre jeu de règles CER avec des non-techniciens ;

- visualisation des dépendances entre vos attributs de règles (notamment au fur et à mesure que vos jeux de règles gagnent en complexité) et
- compréhension des impacts liés aux changements que vous apportez à une dérivation d'attribut de règle.

3.10.1 Exemple simple

Voici le fichier XML d'un jeu de règle simple :

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

Figure 10. Jeu de règles Hello World

Voici le RuleDoc généré pour le jeu de règles ci-dessus, qui répertorie sa classe de règles simple :



Figure 11. RuleDoc HelloWorldRuleSet

Cliquez sur la classe de règles HelloWorld pour afficher le RuleDoc correspondant :



Figure 12. RuleDoc de la classe de règles HelloWorld

Cliquez sur l'attribut greeting pour accéder à sa dérivation :



Figure 13. RuleDoc de l'attribut de règle *greeting*

3.10.2 Exemple plus pratique

Dans les jeux de règles plus complexes, RuleDoc peut vous aider à :

- parcourir les dépendances entre les classes de règles de votre jeu de règles ;
- comprendre le calcul de la dérivation de chaque attribut de règle ; et
- déterminer les autres attributs de règle qui dépendent d'un attribut de règle particulier.

Voici le fichier XML d'un jeu de règles plus complexe :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="RetirementYearRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="RetirementYear">

    <Attribute name="yearOfBirth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="1970"/>
      </derivation>
    </Attribute>

    <Attribute name="ageAtRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="65"/>
      </derivation>
    </Attribute>

    <Attribute name="yearOfRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <reference attribute="yearOfBirth"/>
          <reference attribute="ageAtRetirement"/>
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

Figure 14. Calculateur d'année de départ à la retraite

Voici le RuleDoc généré pour le jeu de règles ci-dessus :



Figure 15. RuleDoc indiquant la dérivation et l'utilisation

Cet exemple indique :

- la dérivation de l'attribut de règle yearOfRetirement (qui lie les attributs de règle yearOfBirth et ageAtRetirement desquels il dépend) ; et
- l'attribut de règle yearOfBirth, avec un lien vers l'attribut de règle yearOfRetirement, dont il dépend directement.

3.10.3 Méthode de génération de RuleDoc

Pour exécuter le générateur de RuleDoc CER, exécutez la commande suivante :

```
build creole.generate.ruledoc
```

La cible exécutera également le valideur de jeu de règles CER sur vos jeux de règles. En cas d'erreurs, le valideur de jeu de règles CER signale les erreurs et les arrêts de traitement. S'il n'y a pas d'erreur, le générateur CER sort RuleDoc pour vos jeux de règles et classes de règles CER.

Conseil : Le valideur de jeu de règles CER signale également les avertissements concernant les problèmes non critiques dans vos jeux de règles. Ces avertissements ne vous empêchent pas d'exécuter et de tester vos règles, mais doivent être traités pour garantir un jeu de règles optimal.

Le générateur de RuleDoc CER place sa sortie dans le répertoire `EJBServer/build/svr/creole.gen/ruledoc`.

3.11 SessionDoc

CER vous permet de publier une documentation HTML relative à une session ("SessionDoc").

Cette documentation fournit un enregistrement de tous les objets de règles créés au cours d'une session, et lorsqu'elle est utilisée en combinaison avec vos tests, elle peut représenter une aide précieuse pour le débogage.

Il peut être utile de recourir au point d'ancrage `tearDown` de JUnit pour forcer toutes les méthodes de test de votre classe de test à élaborer leur documentation SessionDoc :

```
@Override
protected void tearDown() throws Exception {
    /*
     * Ecriture de SessionDoc, sur un répertoire nommé d'après la méthode
     * de test.
     */
    final File sessionDocOutputDirectory =
        new File("./gen/sessiondoc/" + this.getName());
    sessionDoc.write(sessionDocOutputDirectory);

    super.tearDown();
}
```

Voici la page SessionDoc principale d'un test `testSelfMadeMillionaireScenario` :

CREOLE Session

Generated: 13-Jul-2012 12:08:08

Session Type

- Recalculation strategy: curam.creole.execution.session.RecalculationsProhibited
- Data storage: curam.creole.storage.inmemory.InMemoryDataStorage
- Rule object factory: curam.creole.execution.session.StronglyTypedRuleObjectFactory

Options

- "Used by" links included: true

Rule Objects (by Rule Set)

- [FlexibleRetirementYearRuleSet](#)

Figure 16. SessionDoc du test testSelfMadeMillionaireScenario

Voici certains détails clés de cette page :

- Date et heure de création de SessionDoc ;
- Stratégies à l'aide desquelles la session a été créée ;
- Liste des jeux de règles dont les objets de règles ont été capturés dans SessionDoc et
- Indication de l'inclusion des liens "utilisé par".

Le fait de cliquer sur le lien pour le jeu de règles FlexibleRetirementYearRuleSet seul affiche ses objets de règles :

FlexibleRetirementYearRuleSet

Generated: 13-Jul-2012 12:08:08

External rule objects

Details	Type	Description	Action
details	FlexibleRetirementYearRuleSet.FlexibleRetirementYear	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	Created during this session

Internal rule objects

Details	Type	Description	Action
---------	------	-------------	--------

Figure 17. Objets de règles pour le jeu de règles FlexibleRetirementYearRuleSet

Cette page affiche :

- Les objets de règles "externes" (d'amorce) créés par le code client au cours de cette session (un seul a été créé lors du test)

- Les objets de règles "internes" créés par les règles au cours de cette session (aucun n'a été calculé pour ce test).

Le fait de cliquer sur le lien des détails de l'objet de règle FlexibleRetirementYear seul affiche sa documentation SessionDoc :

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by										
ageAtRetirement	Number	CALCULATED	50	<table border="1"> <thead> <tr> <th>If</th> <th>Then</th> </tr> </thead> <tbody> <tr> <td>• retirementCause ==</td> <td></td> </tr> <tr> <td>• "Lottery winner"</td> <td>• 35</td> </tr> <tr> <td>• "Self-made millionaire"</td> <td>• 50</td> </tr> <tr> <td>• Otherwise</td> <td>• 65</td> </tr> </tbody> </table>	If	Then	• retirementCause ==		• "Lottery winner"	• 35	• "Self-made millionaire"	• 50	• Otherwise	• 65	• retirementCause	• yearOfRetirement
If	Then															
• retirementCause ==																
• "Lottery winner"	• 35															
• "Self-made millionaire"	• 50															
• Otherwise	• 65															
description	Message	CALCULATED	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	• Default rule object description.	None	None										
retirementCause	String	SPECIFIED	Self-made millionaire	• Specified externally.	None	• ageAtRetirement										
yearOfBirth	Number	SPECIFIED	1980	• Specified externally.	None	• yearOfRetirement										
yearOfRetirement	Number	CALCULATED	2030	<ul style="list-style-type: none"> • Arithmetic: <ul style="list-style-type: none"> ○ yearOfBirth ○ + ○ ageAtRetirement 	<ul style="list-style-type: none"> • ageAtRetirement • yearOfBirth 	None										

Figure 18. SessionDoc de l'objet de règle FlexibleRetirementYear

En haut (non affiché), on trouve les détails récapitulatifs de l'objet de règle, puis chaque attribut de règle figurant sur l'objet de règle est répertorié, avec les détails suivants :

- **Nom**

Nom de l'attribut de règle

- **Type déclaré**

Type de l'attribut de règle tel qu'il est déclaré dans le jeu de règles. La valeur d'exécution réelle peut être issue d'un sous-type de ce type déclaré

- **Etat**

Etat de la valeur, à savoir :

- **CALCULEE**

Calculée par les règles

- **SPECIFIEE**

Explicitement spécifiée par le code client, remplaçant tout calcul défini, ou initialisée² comme faisant partie d'une expression create

- **NON_ENCORE_CALCULEE**

2. Avant Cúram V6, l'état INITIALISEE était utilisé. A partir de Cúram V6, l'état SPECIFIEE est utilisé en remplacement.

Non spécifiée explicitement, non calculée au cours de l'exécution des règles (car la valeur n'a jamais été demandée par d'autres calculs ou tests) ou

- **ERREUR**

Erreur survenue lors du calcul de la valeur (voir les journaux d'application ou la sortie de la console pour connaître les détails et la pile de calcul de l'erreur).

- **Valeur**

Représentation de l'affichage de la valeur. Si la valeur n'a jamais été calculée (NON_ENCORE_CALCULEE) ou est erronée (ERREUR), alors «?» s'affiche. Si la valeur est un objet de règle, elle s'affiche en tant que lien hypertexte navigable de telle sorte que vous pouvez voir les détails de cet objet de règle et

- **Dérivation**

Dérivation RuleDoc de l'attribut (sans aucun lien). Pour plus d'informations sur RuleDoc, voir 3.10, «RuleDoc», à la page 20.

- **Dépend de**

Liens vers les attributs qui ont été utilisés pour calculer cette valeur.

- **Utilisé par**

(Facultatif) Liens vers les attributs qui utilisaient cette valeur lors du calcul de leurs valeurs.

Conseil : L'implémentation d'un attribut de règle description sur chaque classe de règles peut faciliter la compréhension de votre documentation SessionDoc. Pour en savoir plus, voir 8.2, «Attribut de règle description», à la page 135.

Si vous exécutez SessionDoc sur une base de données étendue, il peut s'avérer utile de supprimer les liens "utilisé par", car leur inclusion risque d'entraîner la génération de très nombreux objets de règles par SessionDoc. Pour supprimer la sortie des liens "utilisé par", utilisez l'élément `curam.creole.execution.session.SessionDoc.write(File, boolean)`, en transmettant *faux* comme deuxième paramètre.

Pour les objets de règles stockés sur les tables de base de données de CER, vous pouvez créer une documentation SessionDoc en exécutant la classe `curam.creole.util.DumpOutRuleObjects`, avec un argument unique correspondant au nom d'un répertoire dans lequel créer la documentation SessionDoc. L'utilitaire `DumpOutRuleObjects` récupère tous les objets de règles à partir des tables de base de données, et l'action relative à chaque objet de règle externe sera donc de type "récupéré". Les éventuels objets de règles internes seront créés (étant donné qu'ils ne sont pas stockés) et afficheront donc une action de type "créé".

Conseil : L'utilitaire `DumpOutRuleObjects` peut représenter un moyen pratique de parcourir les objets de règles stockés sur les tables de base de données de CER. Il peut également constituer une aide précieuse pour le débogage une fois que vous arrivez à la phase d'intégration des règles CER dans votre application en ligne.

Vous pouvez parcourir les objets de règles pour visualiser les valeurs des attributs calculés sur les objets de règles, ainsi qu'une vue technique de la méthode utilisée pour obtenir le résultat d'un calcul donné.

3.12 Attributs de règles inutilisés

CER inclut la prise en charge de la production de rapports relatifs aux attributs de règles qui ne sont pas référencés à partir d'autres calculs dans votre jeu de règles et sont donc candidats à la suppression.

Pour exécuter le rapport des attributs inutilisés de CER, exécutez la commande suivante :

```
build creole.report.unused.attributes
```

CER valide vos jeux de règles et rapporte les attributs de règles inutilisés dans la console.

avertissement : Remarque : Il est tout à fait possible qu'un attribut de règle soit un attribut de règle de "niveau supérieur" qui est référencé uniquement depuis le code client. Un attribut de ce type peut être rapporté comme "inutilisé" par ce rapport, toutefois, un attribut de règle apparemment inutilisé ne doit en aucun cas être supprimé de votre jeu de règles, sauf si vous êtes certain qu'aucun code client ou test ne dépend de cet attribut.

3.13 Consolidateur de jeu de règles CER

Dans Cúram V5.2, CER permettait de diviser votre jeu de règles en plus petits fichiers afin de faciliter le développement simultané de jeux de règles entre plusieurs documents.

Voir A.3, «Instruction Include», à la page 158 pour en savoir plus sur la méthode de division de votre jeu de règles.

Avant le chargement de votre jeu de règles CER sur les données, celui-ci sera automatiquement "consolidé" dans un fichier de jeu de règles unique par les scripts de génération de l'application (en particulier dans la cible **build creole consolidate.rulesets**).

Remarque : Le consolidateur de jeu de règles CER réduit uniquement les instructions Include qui contiennent un emplacement RelativePath.

Tous les autres types d'instructions Include restent volontairement inchangés dans la sortie consolidée.

3.13.1 Exemple

Voici un jeu de règles CER qui en contient un autre :

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Cette classe de règles est définie directement dans ce jeu de règles -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Inclusion d'un jeu de règles défini dans un autre fichier.

  Lorsqu'ils sont assemblés dans un seul jeu de règles, les
  noms de toutes les classes de règles doivent être uniques. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Voici le même jeu de règles après consolidation :

```

<?xml version="1.0" encoding="UTF-8"?><RuleSet
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="Example_Include" xsi:noNamespaceSchemaLocation=
"http://www.curamssoftware.com/CreoleRulesSchema.xsd">

  <!-- Cette classe de règles est définie directement dans ce jeu de règles -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Inclusion d'un jeu de règles défini dans un autre fichier.

  Lorsqu'ils sont assemblés dans un seul jeu de règles, les
  noms de toutes les classes de règles doivent être uniques. -->

  <!--Début de l'inclusion de ./HelloWorld.xml-->
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>
  <!--Fin de l'inclusion de ./HelloWorld.xml-->

</RuleSet>

```

Chapitre 4. Gestion des données

4.1 Introduction

CER effectue des calculs sur les données afin de trouver des résultats qui sont eux-mêmes des données. Depuis Cúram V6, CER prend en charge le stockage des données d'objet de règle dans la base de données et dans des images instantanées.

Ce chapitre décrit la manière dont CER gère les données.

4.2 Sessions CER

Les éléments de données principaux utilisés avec CER sont les suivants :

- **Objets de règle**
Un objet de règle est une instance de classe de règles issue d'un jeu de règles CER ; par exemple, l'objet de règle de la personne John Smith ; et
- **Valeurs d'attribut**
Une valeur d'attribut est la valeur d'un attribut de règle CER sur un objet de règle particulier ; par exemple, la date de naissance de John Smith.

Toutes les interactions avec les objets de règle CER et les valeurs d'attribut apparaissent au sein d'une session CER. Ces interactions incluent la création, l'extraction et/ou la suppression d'objets de règle CER, et tout calcul ou recalcul d'attributs sur les objets de règle.

Chaque session CER peut être créée à l'aide de la classe `curam.creole.execution.session.Session_Factory`.

Lorsqu'une session CER est créée, vous devez indiquer les informations suivantes :

- **Stratégie de recalcul**
La stratégie de traitement d'une demande visant à recalculer une valeur d'attribut CER dans une session CER ³.
La stratégie de traitement d'une demande visant à recalculer une valeur d'attribut CER. Par exemple, le recalcul immédiat, le report du recalcul vers une autre transaction ou le refus de celui-ci.
- **Stockage de données**
Le mécanisme de stockage ⁴à utiliser pour les objets de règle permanents ; par exemple, mémoire en lecture seule (qui sera perdue lorsque la session sort de la portée), stockage de base de données ou stockage par images instantanées.

3. La fonction de CER permettant d'effectuer des recalculs directement est désormais remplacée par le gestionnaire de dépendance (voir Chapitre 5, «Gestionnaire de dépendance», à la page 77).

L'interface et les implémentations de la stratégie de recalcul de CER sont fournies uniquement à des fins de compatibilité amont.

4. Depuis Cúram V6, CER offre un choix d'implémentations de stockage de données. Celles-ci déterminent si les objets de règle, créés ou changés dans une transaction, peuvent être extraits et/ou changés dans une autre transaction.

Il est important de noter que le choix du stockage de données n'a *pas* d'incidence sur la sémantique de vos expressions de règle. Cela signifie que vous pouvez utiliser un stockage de données léger (en mémoire) pour vos tests JUnit (de sorte qu'un grand nombre de tests JUnit puisse s'exécuter rapidement), et un stockage de données permanent (base de données) pour votre logique de production (de sorte que les objets de règle soient conservés pour toutes les transactions), *sans* aucune différence dans vos calculs sous-jacents.

Les implémentations de stockage de données doivent également spécifier une fabrique d'objets de règle à utiliser. Cette fabrique détermine si les objets de règle sont créés selon une méthode fortement typée ou en interprétation seule.

L'application comprend les implémentations suivantes :

- Stratégie de recalcul :
 - **curam.creole.execution.session.RecalculationsProhibited**
Génère une erreur si vous tentez de lancer un calcul au sein de la session CER.
 - **curam.core.sl.infrastructure.propagator.impl.ImmediateRecalculationStrategy**
Exécute immédiatement les recalculs (de manière synchrone, au sein de la même transaction de base de données).
 - **curam.core.sl.infrastructure.propagator.impl.DeferredRecalculationStrategy**
Reporte les recalculs (pour les valeurs d'attribut stockées) à une autre transaction de base de données.
Reporte les recalculs (pour les valeurs d'attribut stockées) à une autre transaction de base de données.
- Stockage de données :
 - **curam.creole.storage.inmemory.InMemoryDataStorage**
Conserve les objets de règle dans la mémoire uniquement. Les objets de règle de ce stockage de données sont disponibles uniquement lorsque le système de stockage de données se trouve dans la portée, en général uniquement pour une transaction de base de données unique.
 - **curam.creole.storage.database.DatabaseDataStorage**
Extrait les objets ⁵règle externes à partir de :
 - **curam.creole.execution.session.RuleObjectsSnapshot.SnapshotDataStorage**
Crée un document XML contenant les détails d'un ensemble d'objets de règle impliqués dans les dépendances d'un ou de plusieurs calculs d'attributs. Fournit une trace d'audit des données finalement utilisés dans ce calcul. En général, le document XML peut être stocké sur une table de base de données de sorte que l'image instantanée des objets de règle puisse être interrogée (mais pas modifiée) par une transaction de base de données ultérieure.
 - un convertisseur d'objet de règle enregistré dans le stockage de la base de données. Chaque convertisseur d'objet de règle désigne la classe de règles qu'il traite, et lorsqu'il est appelé, ⁶le convertisseur d'objet de règle lit les tables métier sous-jacentes pour obtenir les données appropriées et alimenter les objets de règle en mémoire et les renvoie au stockage de données de base de données ; ou
 - Les tables de base de données de CER, destinées au stockage des objets de règle, si aucun convertisseur d'objet de règle n'est enregistré pour traiter la classe de règles requise.

5. Pour une utilisation optimale, seuls les objets de règle externes et leurs valeurs d'attribut sont extraits des données de la base de données Cúram. Les objets de règle internes et leurs attributs, de par leur nature, peuvent être recalculés de manière fiable par la suite, alors que les objets de règle externes contiennent généralement des données provenant de sources externes, et ne peuvent donc pas être recalculés.

6. Notez que chaque convertisseur d'objet de règle est autorisé à imposer des limites à son support pour les expressions A.6.17.53, «readall», à la page 215 et A.6.17.53, «readall», à la page 215.

Par exemple, il se peut que certains convertisseurs d'objet de règle ne prennent pas en charge l'exécution d'une expression A.6.17.53, «readall», à la page 215 (sans un élément imbriqué match), et/ou placent des restrictions sur les attributs de règle pouvant être nommés dans la valeur retrievedattribute de ce match.

Toute violation des limites du convertisseur d'objet de règle du convertisseur entraîne la génération d'une exception lors de l'exécution. Vous devez vérifier que vos tests de jeu de règles incluent une logique qui appelle les convertisseurs d'objet de règle (c'est-à-dire qui s'exécute sur le stockage de données de base de données), contrairement à la majorité de vos tests de logique de règles qui utilise le stockage de données en mémoire (et donc qui n'appelle pas les convertisseurs d'objet de règle).

Consultez la documentation de chaque implémentation du convertisseur d'objet de règle pour comprendre les limites qu'il impose sur son support pour A.6.17.53, «readall», à la page 215.

Les objets de règle externes sont disponibles pour extraction et manipulation par les transactions de base de données suivantes.

- **curam.creole.storage.hybrid.HybridDataStorage**
Combine les aspects comportementaux de InMemoryDataStorage et DatabaseDataStorage. *Réservé à l'usage interne de Cúram uniquement.*
- Fabrique d'objets de règle :
 - **curam.creole.execution.session.StronglyTypedRuleObjectFactory**
Crée et extrait les objets de règle comme des instances de classes Java générées par le générateur de code de test CER (voir 3.6, «Générateur de code de test CER», à la page 16). *Ne doit pas être utilisé dans le code de production.*
 - **curam.creole.execution.session.InterpretedRuleObjectFactory**
Crée et extrait les objets de règle selon une méthode d'interprétation complète (et donc dynamique). (voir 3.5, «Interpréteur de jeu de règles», à la page 13).

ATTENTION :

En règle générale, vous ne devez pas utiliser plus d'une session CER au sein d'une transaction de base de données. Les copies en mémoire des objets de règle d'une session CER sont indépendantes des copies en mémoire des objets de règle de toutes les autres sessions CER.

Le comportement n'est pas garanti si plus d'une session CER (dans la même transaction) tentent d'extraire ou d'interroger le même objet de règle de la base de données.

- **Tests d'unité**

Utilisez InMemoryDataStorage (pour sa vitesse d'exécution) avec StronglyTypedRuleObjectFactory (pour pouvoir utiliser des classes Java générées dans les tests), et RecalculationsProhibited (de sorte que les tests ne changent pas accidentellement les données au cours du processus).

- **Logique de production avec des jeux de règles dynamiques**

Utilisez DatabaseDataStorage (de sorte que les objets de règle soient disponibles pour toutes les transactions) avec InterpretedRuleObjectFactory (pour que les jeux de règles soient entièrement dynamiques), et RecalculationsProhibited, et utilisez les fonctions fournies par le gestionnaire de dépendance (voir Chapitre 5, «Gestionnaire de dépendance», à la page 77) pour effectuer toutes les demandes de recalcul de valeurs CER dans une nouvelle session CER (indépendante).

- **Images instantanées**

Utilisez SnapshotDataStorage (pour que les objets de règle soient lus dans un document XML inaltérable) avec InterpretedRuleObjectFactory (pour que les jeux de règles soient entièrement dynamiques) et RecalculationsProhibited (les images instantanées ne prennent pas en charge les changements).

4.3 Objets de règle externes et internes

CER permet de créer des objets de règle de différentes manières :

- **Externe**

Les objets de règle créés par les clients de CER, hors du contexte d'un calcul. Les objets de règle externes ont tendance à représenter des concepts réels ou d'organisme tels qu'une personne ou un dossier.

- **Interne**

Objets de règle créés par des règles CER (ou dans un code Java appelé par des règles CER). Les objets de règle internes ont tendance à être des éléments calculateurs ou des données dérivées pour une étape intermédiaire dans une chaîne complexe de calculs.

La distinction entre les objets de règle externes et internes est expliquée ci-dessous, et détermine :

- si un objet de règle peut être extrait à l'aide de l'expression A.6.17.53, «readAll», à la page 215 ; et

- si un objet de règle peut être stocké dans la base de données pour extraction lors de transactions ultérieures.

4.3.1 Objets de règle externes

CER permet au code client de poser des questions à un objet de règle (et CER exécute des règles permettant de fournir les réponses à ces questions).

Toutefois, pour que le code client pose une question à un objet de règle, celui-ci doit être connu à la fois par le code client et CER ; en tant que telle, la session de règles CER doit comporter au moins un objet de règle d'amorce créé ou extrait par le code client. Ce code client peut être du code de test ou du code qui intègre CER à une application.

Un objet de règle externe est le point de départ du code client permettant de poser des questions ; cependant, la réponse à cette question peut fournir un objet de règle ou une liste d'objets de règle qui sont soit créés à partir de règles, soit extraits d'autres objets de règle externes.

Important : Une fois que les calculs ont commencé, la stratégie `RecalculationsProhibited` empêche la création d'autres objets de règle, qui rendrait non valides les calculs de A.6.17.53, «readall», à la page 215 effectués précédemment.

Pour éviter ce type d'erreur, vous devez structurer votre code client ou vos tests de sorte que la création de tous les objets de règle de test se produise *avant* les calculs (c.-à-d. avant toute exécution de `getValue`).

4.3.1.1 Exemple

Voici un exemple de jeu de règles :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_externalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Ces attributs doivent être spécifiés lors de la création -->
    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

      <Attribute name="lastName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <!-- Lisez tous les objets de règle de
              type Income -->
        <readall ruleclass="Income"/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Income">
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

Dans le jeu de règles ci-dessus, l'expression A.6.17.53, «readall», à la page 215 est utilisée pour extraire toutes les instances de la classe de règles Income.

Pour créer un objet de règle externe, votre code client ou les tests doit indiquer la session lors de la création de l'objet de règle :

```

package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.RuleObject;
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.parser.RuleSetXmlReader;
import
    curam.creole.ruleclass.Example_externalRuleObjects.impl.Income;
import
    curam.creole.ruleclass.Example_externalRuleObjects.impl.Income_Factory;
import
    curam.creole.ruleclass.Example_externalRuleObjects.impl.Person;
import
    curam.creole.ruleclass.Example_externalRuleObjects.impl.Person_Factory;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

/**
 * Teste les objets de règle externes créés directement par le code client. */
public class TestCreateExternalRuleObjects extends TestCase {

    /**
     * Exemple illustrant la création d'objets de règle externes à l'aide de
     * code généré. */
    public void testUsingGeneratedTestClasses() {

        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        /**
         * Notez que le compilateur impose que le type approprié
         * d'arguments d'initialisation soit fourni.
         */
        final Person person =
            Person_Factory.getFactory().newInstance(session, "John",
                "Smith");
        CREOLETestHelper.assertEquals("John", person.firstName()
            .getValue());

        /**
         * Ces objets sont extraits par l'expression
         * <readall ruleclass="Income"/>
         *
         * dans le jeu de règles. */
        final Income income1 =
            Income_Factory.getFactory().newInstance(session);
        income1.amount().specifyValue(123);

        final Income income2 =
            Income_Factory.getFactory().newInstance(session);
        income2.amount().specifyValue(345);

    }

    /**
     * Exemple illustrant la création d'objets de règle externes à l'aide de
     * l'interpréteur de jeu de règles CER.
     */
    public void testUsingInterpreter() {

36  IBM Curam Social Program Management: Curam Express Rules Reference Manual
    /* read in the ruleSet */
    final RuleSet ruleSet = getRuleSet();

    /* start an interpreted session */

```

4.3.1.2 Quand utiliser les objets de règle externes

Vous devez créer des objets de règle externes pour :

- des objets de règle d'amorce ou de niveau supérieur qui doivent toujours exister pour que votre code client pose des questions pertinentes. Ces objets de règle sont généralement des singletons (c.-à-d. la seule instance de la classe de règle spécifique pendant la session) ; et
- des objets de règle qui sont créés en fonction de certaines données externes (tel qu'une personne ou un dossier).

4.3.2 Objets de règle internes

CER permet aux règles de créer de nouveaux objets de règle, en tant que résultat ou dérivé de l'exécution des calculs.

Pour créer un objet de règle, utilisez l'expression A.6.17.12, «create», à la page 182, en indiquant les valeurs d'argument d'initialisation requises par la classe de règles et/ou les valeurs supplémentaires à spécifier sur l'objet de règle créé.

Important : Les objets de règle créés à l'aide de l'expression A.6.17.12, «create», à la page 182 *ne peuvent pas* être extraits à l'aide des expressions A.6.17.53, «readall», à la page 215, car CER ne peut pas garantir que tous les objets de règle internes ont été ou seront créés (cela dépend de la présence d'une expression A.6.17.12, «create», à la page 182 dans le chemin d'exécution d'un calcul).

4.3.2.1 Exemple

Voici un exemple de jeu de règles CER qui utilise l'expression A.6.17.12, «create», à la page 182 pour créer des objets de règle conditionnels à partir de règles :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_internalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Utilise <create> pour obtenir un nouvel objet de règle.

      D'autres calculs (sur cet objet de règle ou d'autres) peuvent
      accéder à cet objet de règle récemment créé en se référant à
      cet attribut, c'est-à-dire

      <reference attribute="minorAgeRangeTest"/> .

      L'objet de règle créé ne peut PAS être extrait à l'aide d'une expression
      <readall>.
    -->
    <Attribute name="minorAgeRangeTest">
      <type>
        <ruleclass name="AgeRangeTest"/>
      </type>
      <derivation>
        <!-- Créer un test de plage d'âges qui vérifie si cette
          personne est âgée de 0 à 17 ans inclus (c'est-à-dire si elle a
          moins de 18 ans).
        -->
        <create ruleclass="AgeRangeTest">
          <this/>
          <Number value="0"/>
          <Number value="17"/>
        </create>
      </derivation>
    </Attribute>

    <!-- Utilise la vérification de plage d'âges pour déterminer si cette personne
      est mineure. -->
    <Attribute name="isMinor">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <reference attribute="isPersonInAgeRange">
          <reference attribute="minorAgeRangeTest"/>
        </reference>
      </derivation>
    </Attribute>

    <!-- Utilise <create> pour obtenir un nouvel objet de règle, au sein d'une
      autre expression.

      Etant donné que le nouvel objet de règle est créé de manière anonyme,
      il n'est pas accessible à tous les autres objets de règle (mais
      il apparaît toujours comme Créé dans tous les SessionDoc.
    -->
    <Attribute name="isOfWorkingAge">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Créer un test de plage d'âges qui vérifie si cette
          personne est légalement autorisée à travailler (c'est-à-dire si elle est âgée
          d'au moins 16 ans et de moins de 65 ans), puis vérifier

```

Remarque : Comme pour toutes les expressions CER, les expressions A.6.17.12, «create», à la page 182 ne sont calculées que si nécessaire. Par exemple, l'objet de règle *TestPlageAgesMineur* n'est créé que si sa valeur ou celle de *estMineur* est demandée par le code client ou un autre calcul.

Dans l'exemple ci-dessus, la valeur *estEnAgeDeTravailler* utilise la technique de création d'un objet de règle anonyme en encapsulant la création d'un objet de règle dans une expression qui fait référence à un attribut sur l'objet de règle récemment créé. Ces objets de règle ne sont pas accessibles à d'autres calculs mais s'affichent toujours dans n'importe quel SessionDoc généré.

Un objet de règle anonyme peut être utile lorsque vous souhaitez accéder aux attributs de règle dans un objet de règle créé, mais il n'est pas nécessaire de mettre cet objet de règle créé à la disposition de tous les autres calculs.

Regroupement d'objets de règle internes en pools

Depuis Cúram V6, CER conserve un pool d'objets de règle internes qui ont été créés au cours d'une session.

Le pool de session est interrogé à chaque fois qu'une expression A.6.17.12, «create», à la page 182 est évaluée. Si un objet de règle avec les mêmes paramètres d'initialisation et/ou spécifiés a déjà été créé, il est réutilisé à partir du pool et non d'un nouvel objet de règle créé.

Cette approche de regroupement en pool améliore l'efficacité dans les situations où de nombreuses instructions A.6.17.12, «create», à la page 182 tentent de créer des objets de règle identiques. L'utilisation d'un objet de règle unique signifie que tous les attributs calculés sur cet objet de règle unique le sont au moins une fois, et évite des calculs identiques se produisant sur de nombreux objets de règle identiques.

La réutilisation des objets de règle regroupés en pool est garantie comme sûre, car les principes de base de CER garantissent que tout calcul dépend uniquement de ses entrées ; des entrées identiques garantissent donc des sorties identiques.

4.3.2.2 Quand utiliser les objets de règle internes

Vous devez utiliser ce mode pour créer des objets de règle conditionnels en fonction de règles, ou lorsque les valeurs d'attribut d'initialisation sont calculées à partir d'autres règles, ou lorsque l'objet de règle n'est qu'une étape intermédiaire dans un calcul.

Pour un calcul très complexe, vous devez vous attendre à avoir un seul objet de règle externe contenant un attribut pour le résultat du calcul, plusieurs objets de règle externes pour les données d'entrée issues de l'extérieur des règles, et éventuellement un très grand nombre d'objets de règle internes pour les étapes de calcul intermédiaires.

Si vous utilisez des objets de règle qui existent toujours et/ou nécessitent d'être accessibles aux expressions A.6.17.53, «readall», à la page 215, vous devez alors envisager de créer des objets de règle externes directement dans votre code.

4.4 Gestion des types de données

Chaque attribut et expression d'un jeu de règles CER renvoie un élément de données (si demandé). CER prend en charge un ensemble flexible de types de données, qui doivent être spécifiés sur chaque attribut et sur certaines expressions de votre jeu de règles.

4.4.1 Types de données pris en charge

CER inclut la prise en charge des types de données suivants :

- Classes de règles ;
- Classes Java ; et

- Tables de codes d'application.

4.4.1.1 Classes de règles

Toute classe de règles CER définie dans votre jeu de règles peut être utilisée comme type de données de ce jeu de règles.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <!-- Le type de cet attribut est une classe de règles
              définie à un autre endroit de ce jeu de règles.      -->
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Héritage

CER prend en charge l'héritage d'implémentation simple pour les classes de règles.

Une classe de règles peut éventuellement spécifier une déclaration *étend* pour créer une sous-classe contenant une autre classe de règles du même jeu de règles.

Une sous-classe de règles hérite des attributs de règle calculés de toutes ses classes ancêtre, et peut éventuellement remplacer n'importe lequel de ces attributs afin de fournir des règles de calcul de dérivation différentes.

Une sous-classe de règles hérite également des attributs de règle initialisés de toutes ses classes ancêtre, et toute expression A.6.17.12, «create», à la page 182 liée à la sous-classe de règles doit indiquer la valeur des attributs initialisés de toutes les classes de règles ancêtre liées à la sous-classe de règles *avant* celles déclarées sur la sous-classe de règles elle-même.

CER autorise un attribut à être déclaré A.6.17.1, «abstract», à la page 165. Chaque classe de règles qui définit ou hérite d'un attribut abstrait (sans le remplacer) doit également être déclarée abstraite. Une classe abstraite ne peut pas être utilisée dans une expression A.6.17.12, «create», à la page 182.

CER permet à une instance d'objet de règle d'une classe de règles d'être renvoyée à chaque fois que l'une de ses classes de règles ancêtre est attendue.

Le valideur de jeu de règles CER signale une erreur si une expression de votre jeu de règles tente de renvoyer une valeur incompatible :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Le valideur de jeu de règles CER insiste pour que cette
    classe de règles soit marquée comme abstraite, car elle contient
    un attribut de règle abstrait. -->
  <Class name="Resource" abstract="true">
    <Initialization>
      <!-- A chaque fois qu'un objet de règle Ressource est créé,
        son propriétaire doit être initialisé.
        Etant donné que la ressource est abstraite, elle ne peut pas être
        utilisée dans une expression <create> ; cela n'est possible que pour les
        sous-classes concrètes. -->
      <Attribute name="owner">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- La valeur monétaire de la ressource. -->
    <Attribute name="value">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Chaque ressource a un montant, mais celui-ci est
          calculé selon une méthode spécifique à la sous-classe. -->
        <abstract/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Un bâtiment est un type de ressource. -->
  <Class name="Building" extends="Resource">
    <!-- L'adresse physique du bâtiment,
      par exemple, 123 Main Street.
      La valeur d'adresse doit être spécifiée
      en complément de l'attribut de règle
      de propriétaire hérité, qui est un
      attribut initialisé de la super-classe
      de règles. -->
    <Initialization>
      <Attribute name="address">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- Les bâtiments constituent une classe
      concrète, donc le valideur de jeu de règles CER
      insiste pour que cette classe
      hérite d'un calcul ou le déclare
      pour tous les attributs de règle abstraits hérités. -->
    <Attribute name="value">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="-">
          <reference attribute="purchasePrice"> </reference>
          <reference attribute="outstandingMortgageAmount"/>
        </arithmetic>
      </derivation>
    </Attribute>
  </Class>

```

La classe de règles racine

Si une classe de règles ne spécifie aucune autre classe de règles à étendre, la classe de règles étend automatiquement la classe de règles racine de CER, qui contient un seul attribut de règle `description`.

L'attribut de règle `description` fournit une description localisable de l'instance d'objet de règle. Les classes de règles peuvent remplacer librement la dérivation du calcul de règle `description` de leurs instances d'objet de règle.

Chaque classe de règles hérite finalement de la classe de règles racine (et contient donc un attribut de règle `description`), de la même manière dont toutes les classes Java héritent finalement de `java.lang.Object`.

L'implémentation par défaut de l'attribut de règle `description`, fournie par la classe de règles racine, utilise l'expression A.6.17.17, «`defaultDescription`», à la page 189.

4.4.1.2 Classes Java

N'importe quelle classe Java sur le chemin d'accès aux classes de votre application peut être utilisée comme type de données dans votre jeu de règles CER.

ATTENTION :

Lorsque vous stockez des objets de règle dans la base de données, vous pouvez uniquement utiliser des types de données pour lesquels il existe un gestionnaire de type enregistré avec CER.

CER comprend les gestionnaires de type de la plupart des types de données utilisés couramment.

Noms de module

Le nom d'une classe Java doit être entièrement qualifié avec son nom de module, sauf pour les classes des modules suivants :

- `java.lang.*`; and
- `java.util.*`.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaClassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <!-- Il n'est pas nécessaire de spécifier le module
          de java.lang.Boolean -->
        <javaClass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <!-- Nom qualifié complet d'une classe Cúram -->
        <javaClass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

Conseil : Vous ne pouvez pas utiliser les types Java primitifs tels que `boolean` dans CER, mais plutôt leurs équivalents de classe (par exemple, `Boolean`).

Objets non modifiables

Un principe fondamental de CER est que chaque valeur, une fois calculée, ne peut pas être changée.

Pour se conformer à ce principe, les classes Java que vous utilisez doivent être *non modifiables*.

avertissement : Si vous utilisez une classe Java *modifiable* en tant que type de données dans votre jeu de règles CER, vous devez vous assurer qu'aucun code Java ne tente de modifier la valeur d'un objet de cette classe Java. CER ne peut pas garantir la fiabilité des calculs si des valeurs sont modifiées en parallèle !

Heureusement, il existe un grand nombre de classes non modifiables qui répondent généralement à la plupart de vos exigences en matière de type de données. En règle générale, il vous suffit de consulter le JavaDoc d'une classe Java pour déterminer si celle-ci est non modifiable.

Voici une liste utile de classes non modifiables qui suffira probablement à répondre à vos besoins :

- `java.lang.String` ;
- `java.lang.Boolean` ;
- Les implémentations de `java.lang.Number` ; dans tous les cas, CER convertit les instances de `Number` sous leur forme numérique (avec l'aide de `java.math.BigDecimal`) avant d'effectuer toute opération d'arithmétique ou de comparaison ;
- Les implémentations de `java.util.List` qui ne prennent *pas* en charge ses opérations facultatives (voir le JavaDoc pour `List`) ;
- `curam.util.type.Date` ;
- Les implémentations de `curam.creole.value.Message` ; et

- curam.creole.value.CodeTableItem.

Héritage

CER reconnaît la hiérarchie d'héritage des classes et des interfaces Java.

CER autorise le renvoi d'une valeur de classe Java à chaque fois que l'un de ses ancêtres de classe ou d'interface Java est attendu :

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaClassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <javaClass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsObject">
      <type>
        <!-- A titre d'exemple, le renvoi de cette
          valeur en tant que java.lang.Object (peu susceptible
          d'être utile dans un jeu de règles
          réel). -->
        <javaClass name="Object"/>
      </type>
      <derivation>
        <!-- OK, car un booléen *IS* un objet. -->
        <reference attribute="isMarried"/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsString">
      <type>
        <javaClass name="String"/>
      </type>
      <derivation>
        <!-- Le valideur de jeu de règles CER signale l'erreur
          ci-dessous (car un booléen *IS NOT* une chaîne) :

          ERROR    Person.isMarriedAsString
          Example_javaClassInheritance.xml (28, 41)
          Child 'reference' returns 'java.lang.Boolean',
          but this item requires a 'java.lang.String'. -->
        <!-- <reference attribute="isMarried"/> -->

        <!-- (Déclaration comme indiqué afin que cet exemple
          soit généré correctement) -->
        <specified/>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>
```

Classes paramétrées

Java 5 introduit la prise en charge des classes paramétrées, et CER vous permet d'utiliser des classes Java paramétrées dans votre jeu de règles.

Les paramètres d'une classe paramétrée ne sont répertoriés que dans la déclaration `<javaclass>` :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaClassParameterized"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="favoriteWords">
      <type>
        <!-- Une liste de chaînes -->
        <javaClass name="List">
          <javaClass name="String"/>
        </javaClass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="luckyNumbers">
      <type>
        <!-- Une liste de nombres -->
        <javaClass name="List">
          <javaClass name="Number"/>
        </javaClass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="children">
      <!-- Une liste d'objets de règle Personne.

          Etant donné que java.util.List peut être paramétré avec
          n'importe quel objet, il est possible d'utiliser une classe de règles en tant que paramètre. -->
      <type>
        <javaClass name="List">
          <ruleClass name="Person"/>
        </javaClass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Les chiens appartenant à cette personne. -->
    <Attribute name="dogs">
      <type>
        <javaClass name="List">
          <ruleClass name="Dog"/>
        </javaClass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Les chats appartenant à cette personne. -->
    <Attribute name="cats">
      <type>
        <javaClass name="List">
          <ruleClass name="Cat"/>
        </javaClass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Tous les animaux appartenant à cette personne. -->
    <Attribute name="pets">

```

CER vous permet d'utiliser n'importe quel type (y compris les objets de règle) de paramètre qui autorisent `java.lang.Object`. CER impose le typage fort même si le paramètre (par exemple, une classe de règles) est défini de manière dynamique. CER reconnaît également la hiérarchie d'héritage des classes de règles lorsque vous décidez si une classe paramétrée peut être attribuée à un autre.

4.4.1.3 Tables de codes

Toute table de codes d'application peut être utilisée en tant que type de données dans votre jeu de règles CER.

Conseil : La table de codes ne doit *pas* nécessairement exister au moment du développement ; si un administrateur utilise l'application en ligne afin de créer une nouvelle table de codes, cette table de codes peut alors être utilisée en tant que type de données dans les jeux de règles CER définis de manière dynamique.

Pour créer une instance d'une entrée de table de codes (c'est-à-dire pour faire référence à un élément particulier de la table de codes), vous pouvez utiliser l'expression A.6.17.8, «Code», à la page 178.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_codetableentryDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <!-- La valeur de cet attribut doit être
          une entrée de la table de codes Gender
          de Cúram. -->
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Utilisez Code pour créer une valeur codetableentry
          pour comparaison. -->
        <equals>
          <reference attribute="gender"/>
          <Code table="Gender">
            <!-- Code issu de la table de codes -->
            <String value="MALE"/>
          </Code>
        </equals>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

4.4.2 Où spécifier des types de données

Chaque A.5, «Attribut», à la page 162 (à la fois calculé et initialisé) doit spécifier son type.

Pour la plupart des expressions, le type peut être fixe (par exemple, l'expression A.6.17.2, «all», à la page 167 renvoie toujours une valeur Boolean) ou peut être induit (par exemple, une A.6.17.54, «reference», à la page 217 renvoie le type déclaré par l'attribut référencé).

Toutefois, pour certaines expressions le type doit être spécifié explicitement. Il s'agit des expressions suivantes :

- A.6.17.5, «call», à la page 173 ; et
- A.6.17.7, «choose», à la page 176.

En outre, l'expression A.6.17.32, «fixedlist», à la page 196 déclare le type d'élément de la liste renvoyé au sein de son instruction listof.

4.5 Gestion des données variables au fil du temps

Depuis Cúram V6, CER prend en charge une fonction puissante nommée Timelines. Une chronologie CER est tout simplement une valeur qui varie au fil du temps ; c'est la simplicité de cette notion qui permet aux chronologies d'être utilisées efficacement dans l'application.

4.5.1 Définition des données chronologiques

Une chronologie est une séquence de valeurs d'un type donné, dans laquelle chaque valeur est en effet à compter d'une date particulière(jusqu'à ce qu'elle soit remplacée par une autre valeur). Pour n'importe quelle date donnée, une chronologie comporte une valeur applicable à cette date.

4.5.1.1 Exemples de données pouvant être modélisées comme des chronologies CER

Pour introduire la notion de chronologie, voici quelques exemples quotidiens de données qui peuvent varier au fil du temps :

- Les revenus totaux d'une personne ont tendance à varier au fil du temps, lorsqu'elle obtient une augmentation ou change d'emploi. Étant donné que les revenus d'une personne à un moment donné peuvent être représentés sous forme de nombre, la variation des revenus de cette personne au fil du temps peut être représentée sous forme de chronologie numérique, que nous nommerons Timeline<Number> pour plus de simplicité ⁷;
- Quels que soient les revenus totaux, l'enregistrement d'emploi d'une personne a tendance à varier au fil du temps lorsque cette personne change d'emploi (ou lorsqu'elle ne travaille pas pendant certaines périodes). Si, à un moment donné, une personne possède au moins un emploi *primary*, l'historique des emplois principaux de cette personne peut être représenté sous forme de Timeline<Employment>, où l'emploi est une classe de règles ou un type Java qui contient les détails de l'emploi et, au cours des périodes sans emploi principal, la valeur de la chronologie est une valeur marqueur spéciale telle que null (pour représenter Pas d'emploi) ;
- Une personne peut détenir un actif dont elle finit par se séparer, par exemple une personne peut acheter puis vendre une voiture. A n'importe quelle date, la personne est ou n'est pas propriétaire de la voiture, ce qui peut être modélisé sous forme de valeur booléenne. Au fil du temps, la situation selon laquelle la voiture a un propriétaire à une date particulière peut être modélisée sous forme de Timeline<Boolean>. La valeur de la chronologie est false avant la date d'achat du véhicule, et true à partir de la date d'achat et jusqu'à la date de vente incluse (ou «jusqu'à nouvel ordre» si la vente de la voiture n'est pas connue).
- De même, une personne a une date de naissance et finalement une date de décès. Les personnes encore vivantes sont associées à un enregistrement de date de décès vide. A n'importe quelle date, la personne est vivante ou décédée, et donc la valeur dérivée de la question «la personne est-elle vivante ?» peut être modélisée sous la forme d'une valeur booléenne. Au fil du temps, la condition selon laquelle la personne est vivante peut être modélisée comme Timeline<Boolean>. La valeur de la chronologie est

7. La notation utilisée dans ce guide est un emprunt intentionnel à celle des génériques de Java.

fa l se avant la date de naissance de la personne, et true à partir de la date de naissance de la personne et jusqu'à sa date de décès incluse (ou «jusqu'à nouvel ordre» si la personne n'est associée à aucune date de décès).

- Un parent peut avoir plusieurs enfants, nés à des dates différentes. A n'importe quelle date particulière, le parent est associé à une liste d'enfants qui sont vivants à cette date, ce qui peut être modélisé sous forme de List<Person>. Au fil du temps, la liste d'enfants change lorsque d'autres enfants naissent ou atteignent l'âge de la maturité (ou, malheureusement, meurent dans l'enfance), ce qui peut être modélisé sous forme de List<Person>< >

Les exemples ci-dessus sont présentés sous forme de graphique dans la figure ci-dessous.

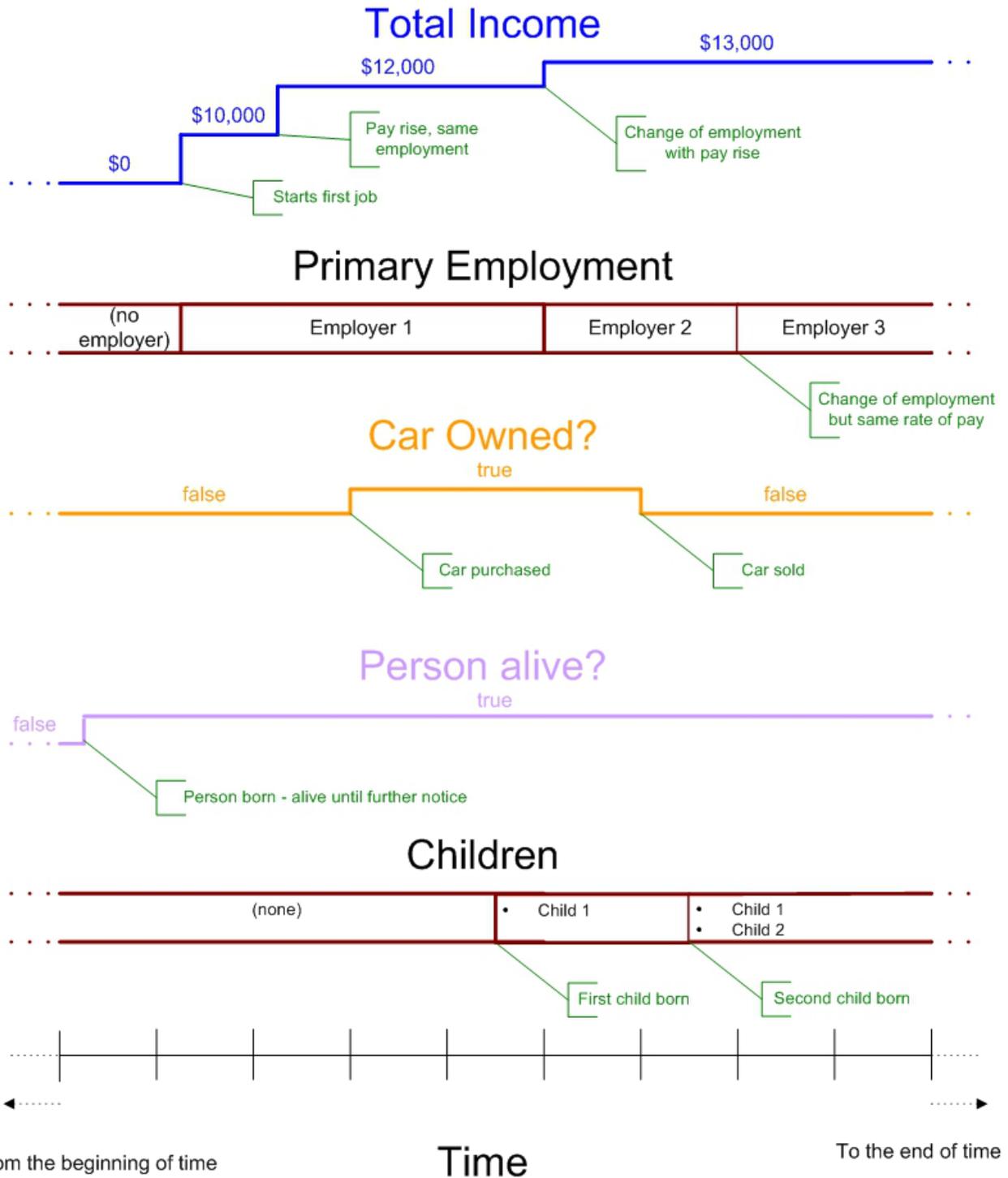


Figure 19. Exemples de données chronologiques

4.5.1.2 Exemples de données non chronologiques

Avant de passer à une étude plus détaillée des chronologies, voici une remarque d'avertissement concernant les données qui ont tendance à ne *pas* être adaptées aux chronologies.

Certains types de données ne sont pas adaptés aux chronologies, car elles ne varient pas avec le temps. Parmi les exemples courants :

- **Identificateurs uniques**

L'une des caractéristiques d'un identificateur unique est qu'il ne varie pas intentionnellement au fil du temps. Par exemple, chaque personne peut être associée à un numéro de sécurité sociale unique. Ce numéro doit être modélisé en tant que nombre, et non `Timeline<Number>`. En revanche, le nom d'une personne peut varier au fil du temps, par exemple suite à un mariage ou à un acte formaliste. C'est l'une des raisons pour lesquelles il n'est pas judicieux de le choisir comme identificateur (avec l'absence d'unicité) ;

- **Dates**

Les données de type date ne varient pas au fil du temps⁸. Par exemple, la date de naissance d'une personne est une date en particulier, et doit donc être modélisée en tant que date, et non en tant que `Timeline<Date>`. Les dates peuvent être utilisées pour *construire* une chronologie (par exemple, les dates de naissance et de décès d'une personne permettent de construire une `Timeline<Boolean>` indiquant si la personne est vivante, mais les dates elles-mêmes ne sont pas des chronologies).

- **Données ponctuelles**

Certaines données capturent intentionnellement les données qui s'appliquent uniquement à une date spécifique. Par exemple, un élément de données `surnameAtBirth` doit être modélisé sous forme de chaîne, et non de `Timeline<String>`. Un élément de données `incomeAtRetirement` doit être modélisé sous forme de nombre, et non de `Timeline<Number>`

ATTENTION :

Lors de la modélisation de données, il est très important d'être clair en indiquant si l'élément de données varie effectivement au fil du temps. N'utilisez la chronologie que pour les éléments de données dont la valeur peut varier au fil du temps.

4.5.2 Comparaison de la chronologie et de perspectives ponctuelles de temps

CER gère les chronologies de façon naturelle, de sorte que lors de la conception de règles CER, vous pouvez faire alterner votre réflexion entre perspective ponctuelle et perspective chronologique.

Les sections suivantes décrivent ces perspectives à l'aide d'exemples. Envisageons d'abord une perspective ponctuelle qui n'implique pas de chronologies. Cet exemple sera réanalysé d'une perspective chronologique.

4.5.2.1 Une perspective ponctuelle

Supposons que nous avons l'exigence métier suivante liée à une dérivation :

8. Dans Cúram, certains éléments de données, tels que les informations collectées, peuvent avoir *deux* sortes d'historique enregistré :

- Un historique de *successions* de données concernant les changements de situation réels, c'est-à-dire les cas où un événement se produit dans le monde réel et signifie que la représentation système de ces données est désormais obsolète, par exemple un changement de revenus de la personne dû à une augmentation ; et
- Un historique de *corrections* des données conservées sur le système, dans lequel il s'avère que le système possède une représentation incorrecte de circonstances réelles ; par exemple, la date de début d'emploi d'une personne a été enregistrée de manière incorrecte.

En tant que tel, pour les éléments de données de type Date, les données peuvent être *corrigées* dans le système, mais jamais *remplacées*. Par conséquent, il peut exister un *historique de corrections* lié à l'élément de données, mais celui-ci (c'est-à-dire les dates auxquelles l'élément de données a été créé) est rarement utile lors de la création de règles CER.

En règle générale, les types de données utilisés dans les règles CER doivent modéliser des circonstances réelles, plutôt que des corrections à la représentation système. Utilisez la chronologie dans les cas où ces circonstances réelles peuvent varier au fil du temps ; et n'utilisez pas la chronologie dans les cas où les données réelles ne peuvent pas varier au fil du temps.

règle : A une date donnée, une personne est considérée comme *chef de famille monoparentale avec mineur à charge* si, à cette date, la personne :

- n'est pas mariée ; et
- s'occupe d'une personne à charge âgée de moins de 16 ans.

A partir de cette exigence simple, il est possible de construire une table de vérité simple pour savoir si une personne est considérée comme chef de famille monoparentale avec mineur à charge, à une date donnée :

		Has a dependent child younger than 16 years of age	
		No	Yes
Is married	Yes	X	✓
	No	✓	X

Is a lone parent of a minor

Figure 20. Table de vérité pour la règle Chef de famille monoparentale avec mineur à charge

Imaginons un scénario de changement de situation réel. Le 1er janvier 2001, Mary et Joe se marient. Joe a un fils, James, issu d'un mariage précédent qui s'est terminé par un divorce le 30 novembre 1998. James est né le 1er juin 1990. Le 30 avril 2004, Joe meurt malheureusement (le mariage de Mary se termine donc par un veuvage).

Vous pouvez utiliser la table de vérité ci-dessus pour déterminer si chacune de ces personnes est considérée comme *chef de famille monoparentale avec mineur à charge* à différentes dates :

- Le 1er octobre 1997 (date choisie au hasard), Mary n'est pas chef de famille monoparentale avec mineur à charge, car à cette date elle n'est pas mariée mais elle n'a aucune personne à charge ;
- Le 2 octobre 1997, Mary n'est toujours pas chef de famille monoparentale avec mineur à charge, car sa situation n'a pas changé depuis la veille ;
- Le 30 novembre 1998, Joe n'est pas chef de famille monoparentale avec mineur à charge, car à cette date son fils avait moins de 16 ans mais Joe était toujours marié ;
- Le 1er décembre 1998, Joe devient chef de famille monoparentale avec mineur à charge, car à cette date son fils avait encore moins de 16 ans mais Joe n'était plus marié ;
- Le 1er janvier 2001, Mary n'est toujours pas chef de famille monoparentale avec mineur à charge ; même si elle s'occupe désormais d'une personne à charge de moins de 16 ans, elle est alors mariée, donc pour des raisons quelque peu différentes de précédemment, elle n'est toujours pas chef de famille monoparentale avec mineur à charge ;

- Le 1er janvier 2001, Joe n'est plus chef de famille monoparentale avec mineur à charge ; bien qu'il s'occupe toujours d'une personne à charge de moins de 16 ans, il est à présent marié ;
- Le 1er mai 2004, Mary devient chef de famille monoparentale avec mineur à charge, car son mariage s'est terminé suite au décès de Joe, mais James est toujours sa personne à charge et est âgé de moins de 16 ans ;
- Le 1er juin 2006, Mary cesse d'être chef de famille monoparentale avec mineur à charge, car James a 16 ans ;
- Le 1er mars 2009 (autre date choisie au hasard), James n'est pas chef de famille monoparentale avec mineur à charge, car même s'il n'est pas marié, il n'a pas de personnes à charge.

Notez que nous devons évaluer la table de vérité pour diverses dates, en vue de constituer une image des moments auxquels chaque personne est ou n'est pas chef de famille monoparentale avec mineur à charge. Dans une certaine mesure, vous devez utiliser des dates qui vous paraissent «intéressantes», ou choisir des dates un peu au hasard. Par exemple, on pouvait penser que la date de mariage de Mary serait intéressante, mais il s'avère que son mariage avec Joe n'a *pas* d'incidence sur son statut de chef de famille monoparentale avec mineur à charge. Toutefois, le statut de chef de famille monoparentale avec mineur à charge de Joe *change* lorsqu'il épouse Mary. Les dates précédant la naissance de James n'ont pas été utilisées pour tester si Joe était alors chef de famille monoparentale avec mineur à charge.

4.5.2.2 Une perspective chronologique

Nous allons maintenant reprendre l'exemple de règle et de situation d'une perspective chronologique.

Tout d'abord, nous pouvons reformuler légèrement l'exigence de la manière suivante :

règle (reformulée) : Une personne est considérée comme *chef de famille monoparentale avec mineur à charge* lorsqu'elle :

- *n'est pas mariée* ; et
- *s'occupe d'une personne à charge âgée de moins de 16 ans.*

(Les expressions «A une date donnée» et «à cette date» ont été supprimées et remplacées par «lorsque». Cette reformulation est mineure, mais peut être essentielle pour passer d'une réflexion axée sur des points dans le temps à une réflexion sur des données variant au fil du temps.)

Nous allons maintenant élaborer des chronologies (de type Timeline<Boolean> pour les périodes auxquelles chaque personne :

- est mariée ; et
- s'occupe d'une personne à charge âgée de moins de 16 ans.

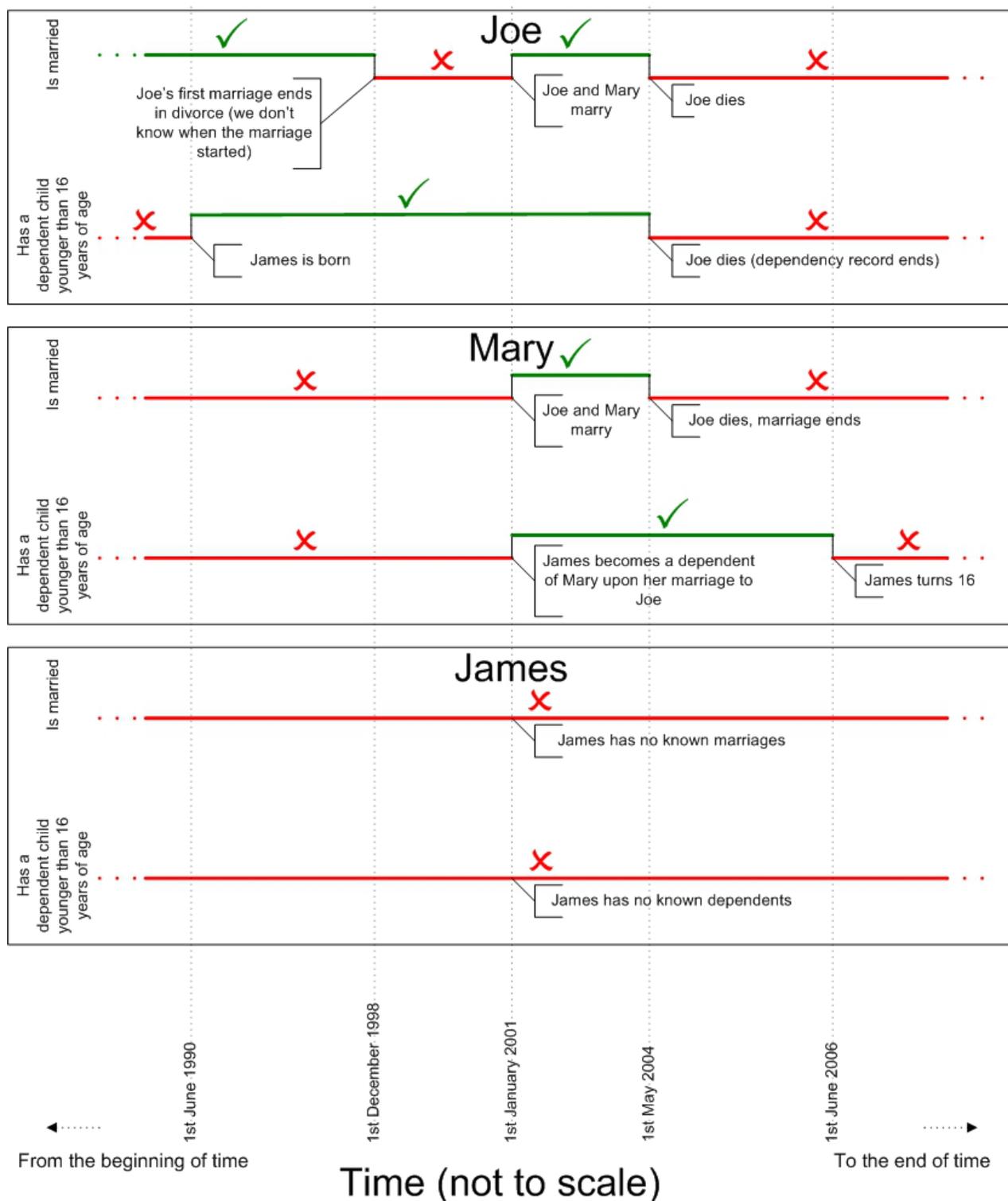


Figure 21. Chronologies des situations de Joe, Mary et James

A partir de ces chronologies des situations de Joe, Mary et James, il est possible d'élaborer de nouvelles chronologies pour en déduire l'évolution dans le temps de leur statut de chef de famille monoparentale avec mineur à charge :

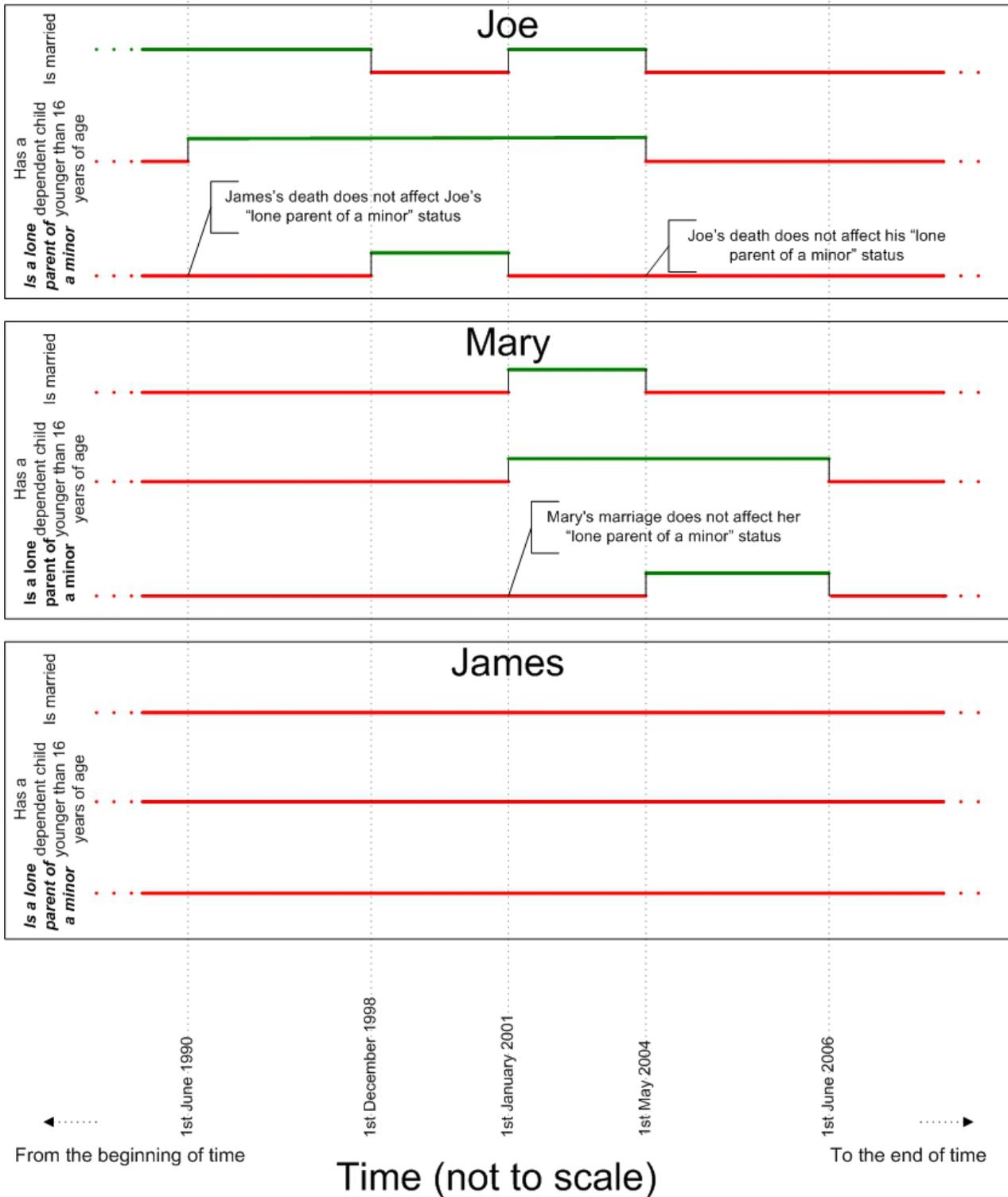


Figure 22. Chronologie du statut de Joe, Mary et James en tant que chef de famille monoparentale avec mineur à charge

Notez qu'il est possible de lire immédiatement l'évolution du statut de chef de famille monoparentale avec mineur à charge pour chaque personne, sans avoir à deviner les dates intéressantes :

- Joe est chef de famille monoparentale avec mineur à charge du 1er décembre 1998 au 31 décembre 2000 inclus ;

- Mary est chef de famille monoparentale avec mineur à charge du 1er mai 2004 au 30 juin 2006 inclus ;
et
- James n'est jamais chef de famille monoparentale avec mineur à charge.

4.5.3 Construction de chronologies

Nous avons vu comment appliquer des expressions à des données de chronologie existante afin de calculer des données constituant une chronologie.

Cette section explique comment les données chronologiques sont créées en premier lieu.

Il existe deux méthodes de création des chronologies :

- en code Java, que ce soit par un client de CER ou dans le cadre d'un code Java appelé lors d'un appel de CER ; et/ou
- dans des règles CER, à l'aide d'expressions CER qui créent des données chronologiques à partir de données primitives (non chronologiques).

4.5.3.1 Construction de chronologies dans un code Java

Dans Java, chaque élément de données chronologiques est une instance de la classe paramétrée `curam.creole.value.Timeline`. Pour des détails complets sur cette classe, reportez-vous à son JavaDoc disponible à l'adresse `EJBServer/components/CREOLEInfrastructure/doc` dans une installation de développement de l'application.

Chaque chronologie contient un ensemble d'intervalles, dans laquelle les *value* d'un intervalle sont applicables à partir d'une *start date* particulière. Un ensemble d'intervalles appropriés doit être transmis au constructeur de la chronologie.

Par exemple, supposez que vous deviez créer une `Timeline<Number>` avec ces intervalles (rappelez-vous qu'une chronologie s'étend indéfiniment loin dans le passé et le futur) :

- 0 jusqu'au 31 décembre 2000 inclus ;
- 10 000 à partir du 1er janvier 2001 jusqu'au 30 novembre 2003 inclus ; et
- 12 000 à partir du 1er décembre 2004 jusqu'à nouvel ordre.

Voici quelques exemples de code Java permettant de créer cette chronologie :

```

package curam.creole.example;

import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class CreateTimeline {

    /**
     * Créer une chronologie numérique avec ces valeurs d'intervalle :
     * <ul>
     * <li>0 jusqu'au 31 décembre 2000 inclus;</li>
     * <li>10 000 entre le 1er janvier 2001 et le 30
     * novembre 2003 inclus; et</li> * <li>12 000 à partir du 1er décembre 2004 jusqu'à nouvel ordre.</li>
     * </ul>
     */
    public static Timeline<Number> createNumberTimeline() {

        return new Timeline<Number>(

            // premier intervalle, application à partir du début de la durée
            new Interval<Number>(null, 0),

            // second intervalle
            new Interval<Number>(Date.fromISO8601("20010101"), 10000),

            // dernier intervalle (jusqu'à nouvel ordre)
            new Interval<Number>(Date.fromISO8601("20041201"), 12000)

        );
    }
}

```

Autre exemple : voici quelques exemples de code Java, que l'on peut appeler une expression CER call, pour calculer une chronologie liée à l'âge d'une personne, jusqu'à son 200^e anniversaire (rappelez-vous que les chronologies d'âge doivent être limitées artificiellement, de sorte que la chronologie contienne un nombre limité de changements de valeur) :

```

package curam.creole.example;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class AgeTimeline {

    /**
     * Crée une chronologie liée à l'âge d'une personne, artificiellement
     * limitée à 200 anniversaires.
     * <p>
     * Peut être appelée à partir de règles CER via une expression &lt;call&gt;.
     */
    public static Timeline<? extends Number> createAgeTimeline(
        final Session session, final Date dateOfBirth) {

        /**
         * La limite artificielle, de sorte que la chronologie d'âge comporte un
         * nombre limité de changements de valeur.
         */
        final int NUMBER_OF_BIRTHDAYS = 200;

        final Collection<Interval<Integer>> intervals =
            new ArrayList<Interval<Integer>>(NUMBER_OF_BIRTHDAYS + 2);

        /**
         * l'âge précédant la date de naissance doit toujours être enregistré à 0 -
         * crée une application d'intervalle initiale à partir du
         * début de la durée
         */
        final Interval<Integer> initialInterval =
            new Interval<Integer>(null, 0);
        intervals.add(initialInterval);

        /**
         * Identifier chaque anniversaire jusqu'à la limite. Notez que la personne
         * le premier anniversaire est fusionné en un seul intervalle par la
         * chronologie, même s'il est plus clair de conserver la logique
         * en l'état.
         */
        for (int age = 0; age <= NUMBER_OF_BIRTHDAYS; age++) {

            // calculer la date d'anniversaire
            final Calendar birthdayCalendar = dateOfBirth.getCalendar();

            /**
             * NB utiliser .roll et non .add pour obtenir un traitement
             * correct pour les années bissextiles
             */
            birthdayCalendar.roll(Calendar.YEAR, age);
            final Date birthdayDate = new Date(birthdayCalendar);

            /**
             * l'âge s'applique à partir de cet anniversaire jusqu'au suivant
             */
            intervals.add(new Interval<Integer>(birthdayDate, age));
        }

        final Timeline<Integer> ageTimeline =
            new Timeline<Integer>(intervals);

        return ageTimeline;
    }
}

```

Remarque : En général, les données chronologiques tendent à être créées en dehors des règles, par les clients de CER.

En particulier, le système de traitement de Cúram V6 Eligibility/Entitlement contient une logique qui permet de convertir les informations collectées de Cúram en données chronologiques.

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules* pour plus de détails.

4.5.3.2 Construction de chronologies dans des règles CER

Généralement, les données chronologiques sont créées en dehors des règles par les clients de CER et utilisées pour remplir la valeur d'un attribut CER à l'aide du mécanisme de spécification.

Toutefois, CER contient également des expressions permettant de créer des chronologies directement dans des règles CER :

- `Timeline`, en association avec `Interval` ; et
- `existencetimeline`.

Chronologie et intervalle

Vous pouvez créer une chronologie de manière native dans des règles CER en créant d'abord explicitement une liste d'intervalles, puis en utilisant cette liste pour créer une chronologie.

Dans la pratique, ces chronologies fixes tendent à être utiles uniquement comme mesure temporaire, lorsque vous étoffez votre jeu de règles.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateTimelines">

    <!-- Cet exemple utilise <initialvalue> pour définir la valeur valide
      à partir du début de la durée. -->
    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <!-- Valeur de début de durée -->
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- Intervalles restants -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2004-12-01"/>
                  </start>
                  <value>
                    <Number value="12000"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>
          </intervals>
        </Timeline>
      </derivation>
    </Attribute>

    <!-- Cet exemple n'utilise pas <initialvalue>. -->
    <Attribute name="aStringTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="String"/>
        </javaclass>
      </type>
    </Attribute>
  </Class>
</RuleSet>

```

existencetimeline

Certains objets métier possèdent des dates de début et de fin naturelles, qui spécifient une période pour laquelle l'objet métier *exists*. L'une ou les deux dates de début et de fin peuvent être facultatives, auquel cas la période d'existence de l'objet métier est à durée indéterminée.

Exemples :

- un emploi, qui démarre et se termine par la suite ;
- un actif, qui est acheté et vendu ultérieurement ; et
- une personne qui est née et mourra un jour.

Les dates de début et de fin d'un objet métier peuvent être utilisées pour diviser une durée en ces trois périodes (ou moins, si la date de début ou la date de fin est vide) :

- **Période de pré-existence**
la période antérieure à la date de début métier (si la date de début existe) ;
- **Période d'existence**
la période comprise entre la date de début métier et la date de fin métier incluse ;
- **Période de post-existence**
la période postérieure à la date de fin métier (si la date de fin existe).

Il peut souvent s'avérer utile d'attribuer une valeur différente à chacune de ces périodes pour un objet métier, et de créer une chronologie à partir de ces valeurs. CER contient une expression `existencetimeline` permettant de créer une chronologie de valeurs de pré-existence/existence/post-existence basée sur des dates de début et de fin facultatives.

Si la date de début n'existe pas, il n'y aura aucun intervalle de pré-existence dans la chronologie. Par exemple, si un actif ne comporte pas de date d'achat enregistrée, sa valeur effective s'applique à partir du début de la durée, sans période de valeur zéro.

Si la date de fin n'existe pas, il n'y aura aucun intervalle de post-existence dans la chronologie. Par exemple, si un actif ne comporte aucune date de vente, alors la valeur de cet actif est valable jusqu'à nouvel ordre (c'est-à-dire arbitrairement à long terme).

Voir 7.4.6, «Existence Timeline», à la page 124 pour plus de détails sur la manière d'utiliser les chronologies d'existence dans l'éditeur CER.

4.5.4 Exploitation des chronologies

Les chronologies CER permettent de stocker les données qui varient au fil du temps. CER prend en charge une fonction `Timeline Operation` qui permet aux expressions CER de fonctionner sur des éléments de données chronologiques pour produire des résultats chronologiques.

4.5.4.1 Conservation des dates de changement

Toutes les expressions peuvent fonctionner sur des chronologies de telle sorte que les dates de changement des valeurs chronologiques d'entrée soient naturellement associées aux dates de changement de la valeur chronologique obtenue.

Les exemples ci-dessus ont introduit la notion d'exploitation des chronologies (*chronologies est marié, a une personne à charge de moins de 16 ans*) pour produire une chronologie de sortie (*chronologie est chef de famille monoparentale avec mineur à charge*).

Plus formellement, pour n'importe quelle expression CER exploitant une ou plusieurs valeurs, CER permet à cette expression de fonctionner également sur une chronologie de ces valeurs. En général, toute exploitation pouvant être appliquée à des types primitifs (par ex. `Date`, `Number`, `String`, `Boolean`, etc.) en

vue d'obtenir un résultat, peut à la place être appliquée à des chronologies appartenant à ces types (par ex. Timeline<Date>, Timeline<Number>, Timeline<String>, Timeline<Boolean>) pour arriver à un résultat qui est une valeur chronologique.

CER contient des expressions spéciales nommées A.6.17.67, «*timelineoperation*», à la page 236 et A.6.17.37, «*intervalvalue*», à la page 199, qui empêchent les autres expressions CER de reconnaître qu'elles fonctionnent sur Timelines.

Par exemple, CER contient une expression A.6.17.64, «*sum*», à la page 230 permettant d'ajouter une liste de nombres. Si une personne possède plusieurs revenus, il est possible d'en faire la somme à un moment donné afin d'obtenir les revenus globaux de la personne à ce moment. Toutefois, dans le cas de *chronologies* présentant l'évolution des montants de ces revenus au fil du temps, il est possible d'utiliser aussi facilement l'expression A.6.17.64, «*sum*», à la page 230 pour obtenir l'évolution des revenus globaux au fil du temps :

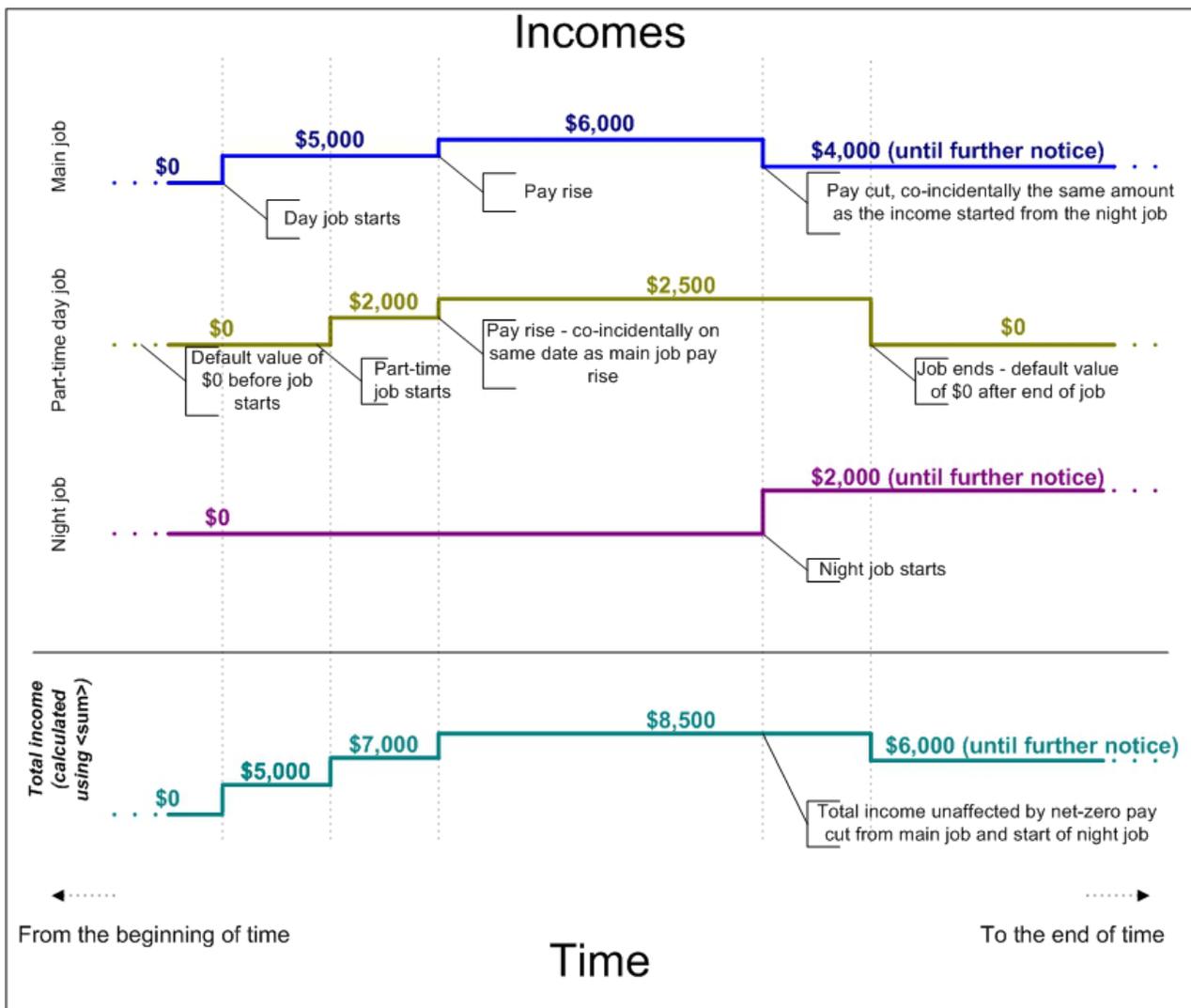


Figure 24. Une chronologie des revenus globaux, calculée à l'aide de *sum*

4.5.4.2 Déplacement de dates

En fonction de vos besoins métier, il peut être souhaitable de créer une chronologie basée sur une autre, dans laquelle les dates de changement de valeur de la chronologie obtenue sont différentes de celles de la chronologie d'entrée.

CER n'inclut pas d'expressions de déplacement de date, car les types de déplacement de date requis ont tendance à être spécifiques au domaine d'activité. L'approche recommandée consiste à créer une méthode Java statique pour créer la chronologie souhaitée et à appeler cette méthode statique à partir des règles à l'aide de l'expression A.6.17.5, «call», à la page 173.

Important : Lorsque vous implémentez un algorithme de déplacement de date, vous devez vérifier qu'il n'existe aucune tentative de création d'une chronologie avec plusieurs valeurs à une date donnée, car ce type de tentative aboutit à un échec lors de l'exécution.

Les tests de votre algorithme doivent inclure tous les cas particuliers, tels que les années bissextiles ou les mois qui comportent un nombre de jours différent.

Exemple d'ajout de date

Votre besoin métier est le suivant : une personne n'est pas autorisée à demander un type de prestation dans les trois mois suivant la réception de ces mêmes prestations.

Pour implémenter cette exigence métier, vous disposez déjà d'une chronologie `isReceivingBenefitTimeline` qui indique les périodes pendant lesquelles une personne reçoit des prestations.

Vous avez désormais besoin d'une autre chronologie `isDisallowedFromApplyingForBenefitTimeline` qui indique les périodes pendant lesquelles toute nouvelle demande de prestations de la part de cette personne n'est pas recevable. Cette chronologie représente un ajout de date de 3 mois aux dates de changement de valeur dans `isReceivingBenefitTimeline` :

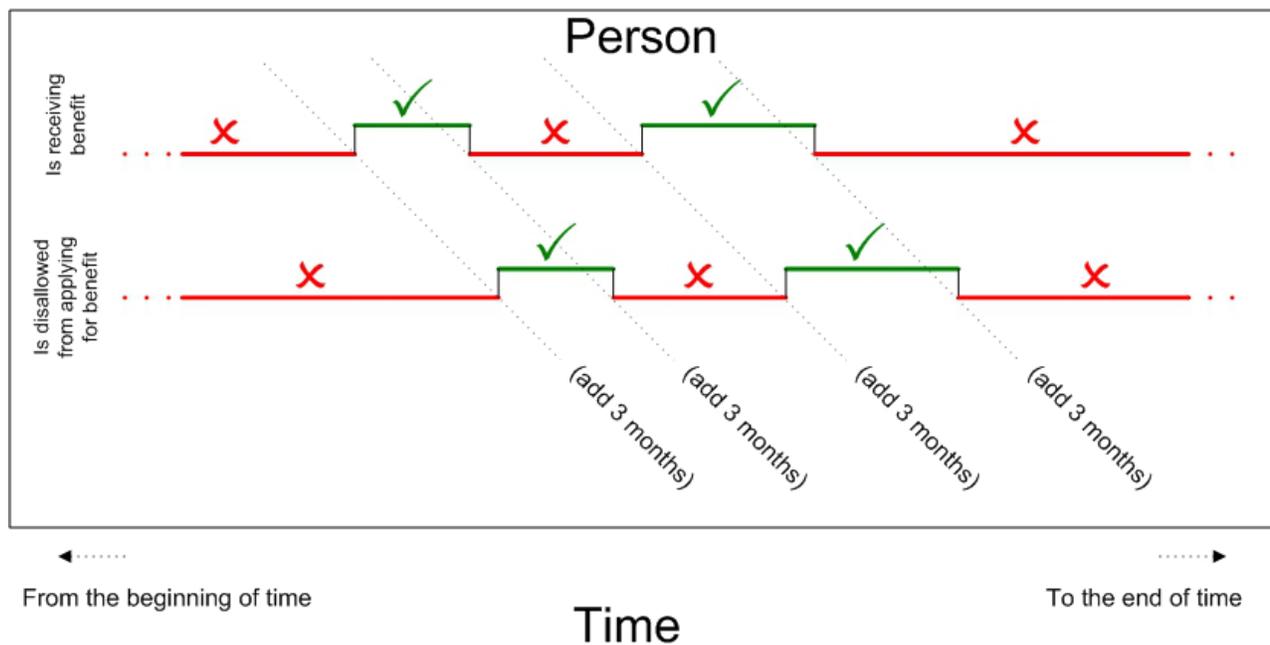


Figure 25. Exigence liée à une chronologie d'ajout de date

Voici un exemple d'implémentation de méthode statique qui peut être appelée à partir de règles CER :

```

package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateAdditionTimeline {

    /**
     * Crée une chronologie en fonction de la chronologie d'entrée, avec la date
     * décalée du nombre de mois indiqué. * <p>
     * Notez que le paramètre de la chronologie peut être de n'importe quel type. *
     * @param session
     * la session CER
     * @param inputTimeline
     * la chronologie dont les dates doivent être décalées
     * @param monthsToAdd
     * le nombre de mois à ajouter à la chronologie de changement
     * dates
     * @param <VALUE>
     * le type de valeur conservée dans les chronologies d'entrée-sortie
     * @return une nouvelle chronologie avec les valeurs de la chronologie
     * d'entrée, décalée du nombre de mois indiqué. */
    public static <VALUE> Timeline<VALUE> addMonthsTimeline(
        final Session session, final Timeline<VALUE> inputTimeline,
        final Number monthsToAdd) {

        /*
         * CER transmet généralement un numéro, qui doit être converti en * nombre entier */
        final int monthsToAddInteger = monthsToAdd.intValue();

        /*
         * Rechercher les intervalles dans la chronologie d'entrée */
        final List<? extends Interval<VALUE>> inputIntervals =
            inputTimeline.intervals();

        /*
         * Cumuler les intervalles de sortie. Notez que le mappage s'effectue par date de début, * car lorsque vous ajou
         *
         * Par exemple, pour calculer 3 mois après ces dates : 2002-11-28, * 2002-11-29, 2002-11-30, on obtient dans tou
         *
         * Dans ce cas, vous devez utiliser la valeur de la première date d'entrée * uniquement - les dates d'entrée son
         */
        final Map<Date, Interval<VALUE>> outputIntervalsMap =
            new HashMap<Date, Interval<VALUE>>(inputIntervals.size());

        for (final Interval<VALUE> inputInterval : inputIntervals) {
            // obtenir la date de début d'intervalle
            final Date inputStartDate = inputInterval.startDate();

            /*
             * Ajouter le nombre de mois - mais n mois après le début de la * durée correspond toujours au début de la
             */

            final Date outputStartDate;
            if (inputStartDate == null) {
                outputStartDate = null;
            } else {
                final Calendar startDateCalendar =
                    inputStartDate.getCalendar();

                startDateCalendar.add(Calendar.MONTH, monthsToAddInteger);
                outputStartDate = new Date(startDateCalendar);
            }

            // vérifier que cette date de sortie n'a pas encore été traitée
            if (!outputIntervalsMap.containsKey(outputStartDate)) {

```

Exemple d'extension de dates

Votre exigence métier est la suivante : une voiture doit être taxée chaque mois où elle «effectue des trajets», sur un ou plusieurs jours de ce mois.⁹

Pour implémenter cette exigence métier, vous disposez déjà d'une chronologie `isOnRoadTimeline` qui indique les périodes pendant lesquelles une voiture «effectue des trajets».

Vous avez désormais besoin d'une autre chronologie `taxDueTimeline` qui indique les périodes pour lesquelles la voiture doit être taxée. Cette chronologie est une extension des dates comprises dans `isOnRoadTimeline` :

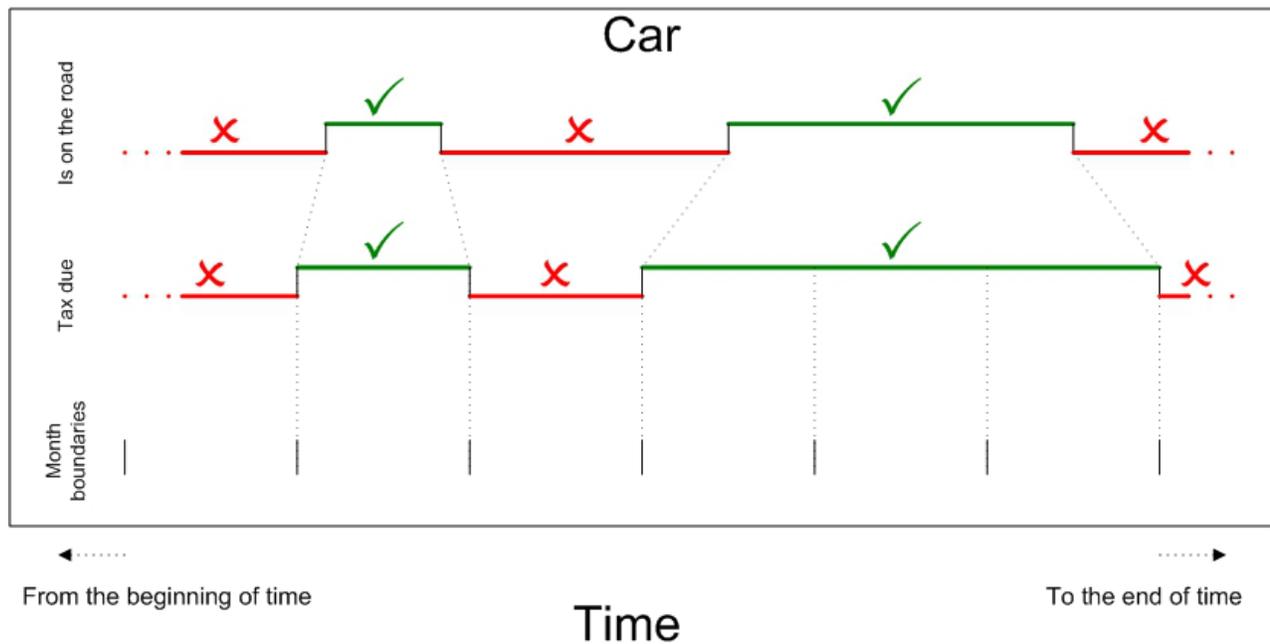


Figure 26. Exigence liée à une chronologie d'extension de dates

Voici un exemple d'implémentation de méthode statique qui peut être appelée à partir de règles CER :

9. Cela signifie que si une voiture effectue d'autres trajets au cours d'un mois, le propriétaire de la voiture doit s'assurer que la taxe est payée rétrospectivement pour le mois complet.

```

package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateSpreadingTimeline {

    /**
     * Crée une chronologie de la période pour laquelle une voiture doit être * taxée.
     * <p>
     * La voiture doit être taxée pour le mois complet, pour n'importe quel mois où * elle effectue des trajets sur un
     */
    public static Timeline<Boolean> taxDue(final Session session,
        final Timeline<Boolean> isOnRoadTimeline) {

        /*
         * Rechercher les intervalles dans la chronologie d'entrée */
        final List<? extends Interval<Boolean>> isOnRoadIntervals =
            isOnRoadTimeline.intervals();

        /*
         * Cumuler les intervalles de sortie. Notez que le mappage s'effectue par date de début ;
         * il se peut qu'une voiture n'effectue aucun trajet pendant un mois, ce qui signifie
         * qu'aucune taxe n'est requise au début du mois suivant, sauf si
         * elle effectue d'autres trajets au cours du mois suivant. Dans ce cas,
         * le versement d'une taxe est finalement requis.
         *
         * Par exemple, la voiture effectue un nouveau trajet le 2001-01-15, la taxe est donc
         * requise (rétrospectivement) à partir du 2001-01-01.
         *
         * Le 2001-01-24, la voiture n'effectue pas de trajet, il est donc
         * possible qu'aucune taxe ne soit imposée à partir du
         * 2001-02-01.
         *
         * Cependant, le 2001-02-05 la voiture effectue un trajet,
         * elle sera donc finalement taxée à partir du 2001-02-01. La
         * chronologie obtenue fusionne ces périodes pour indiquer que la
         * voiture doit être taxée à partir du 2001-01-01 (en couvrant également la période
         * débutant le 2001-02-01).
         */
        final Map<Date, Interval<Boolean>> taxDueIntervalsMap =
            new HashMap<Date, Interval<Boolean>>(
                isOnRoadIntervals.size());

        for (final Interval<Boolean> isOnRoadInterval :
            isOnRoadIntervals) {
            // obtenir la date de début d'intervalle
            final Date isOnRoadStartDate = isOnRoadInterval.startDate();

            if (isOnRoadStartDate == null) {
                // au début de la durée, la voiture doit être taxée si elle effectue
                // des trajets
                taxDueIntervalsMap.put(null, new Interval<Boolean>(null,
                    isOnRoadInterval.value()));
            } else if (isOnRoadInterval.value()) {
                /*
                 * début d'une période de trajet de la voiture - la voiture
                 * doit être taxée à partir du début du mois contenant le
                 * début de cette période */

                final Calendar carOnRoadStartCalendar =
                    isOnRoadStartDate.getCalendar();
                final Calendar startOfMonthCalendar =
                    new GregorianCalendar(

```

4.5.5 Test des chronologies de sortie

Un attribut de règle qui renvoie une chronologie de valeurs doit être testé dans vos tests JUnit de manière similaire aux tests de valeurs primitives (non chronologiques).

Pour simplifier vos tests, vous n'avez pas besoin de vérifier que A.6.17.67, «*timelineoperation*», à la page 236 cumule correctement les dates de changement (sauf si vous le souhaitez). Pour ne pas compliquer vos tests, vous pouvez généralement utiliser des chronologies d'entrée dont la valeur reste constante.

Par exemple, imaginons que vous disposez d'un attribut de règle qui calcule le total (de manière chronologique) à partir d'une liste de nombres :

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_NumberSumTimeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Totalizer">

    <!-- Les chronologies à cumuler -->
    <Attribute name="inputNumberTimelines">
      <type>
        <javaclass name="List">
          <javaclass name="curam.creole.value.Timeline">
            <javaclass name="Number"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Le total obtenu -->
    <Attribute name="totalTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <timelineoperation>
          <sum>
            <dynamiclist>
              <list>
                <reference attribute="inputNumberTimelines"/>
              </list>
              <listitemexpression>
                <intervalvalue>
                  <current/>
                </intervalvalue>
              </listitemexpression>
            </dynamiclist>

            </sum>
          </timelineoperation>
        </derivation>
      </Attribute>

    </Class>
  </RuleSet>
```

...vous pouvez alors écrire un test simple qui utilise des chronologies d'entrée dont la valeur reste constante :

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Timeline;

public class TestForeverValuedTimelines extends TestCase {

    public void testNumberSumTimeline() {

        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        final Totalizer totalizer =
            Totalizer_Factory.getFactory().newInstance(session);

        // utiliser des valeurs d'entrée qui ne varient pas au fil du temps
        final Timeline<Number> inputTimeline1 =
            new Timeline<Number>(1);
        final Timeline<Number> inputTimeline2 =
            new Timeline<Number>(2);
        final Timeline<Number> inputTimeline3 =
            new Timeline<Number>(3);

        totalizer.inputNumberTimelines().specifyValue(
            Arrays.asList(inputTimeline1, inputTimeline2,
                inputTimeline3));

        // vérifier que la chronologie obtenue est toujours égale à 6
        CREOLETestHelper.assertEquals(new Timeline<Number>(6),
            totalizer.totalTimeline().getValue());
    }
}
```

Conseil : La classe `Timeline` comprend un constructeur de commodité permettant de créer une chronologie dont la valeur reste constante.

Dans certains cas, par exemple lorsque vous avez créé votre propre algorithme de déplacement de date, ou que vous avez vraiment besoin de vérifier que les dates de changement des chronologies d'entrée sont répercutées correctement dans les chronologies obtenues, il existe différentes approches que vous pouvez adopter selon vos besoins :

- **Contrôle strict**

Vous devez vérifier que la chronologie obtenue est exactement égale à une valeur chronologique attendue. (La sémantique d'égalité de la classe `Timeline` se comporte de manière prévisible : deux

chronologies sont égales si elles contiennent exactement le même ensemble d'intervalles, c'est-à-dire si les valeurs issues de ces deux chronologies sont identiques pour chaque date possible).

- **Contrôle rapide**

Vous pouvez vérifier que la chronologie obtenue comporte la valeur que vous attendez à des dates particulières.

```

package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class TestStrictTimelineChecking extends TestCase {

    public void testNumberSumTimeline() {
        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        final Totalizer totalizer =
            Totalizer_Factory.getFactory().newInstance(session);

        // utiliser des valeurs d'entrée qui varient au fil du temps
        final Timeline<Number> inputTimeline1 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 1),
                new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
            ));

        final Timeline<Number> inputTimeline2 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 2),
                new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
            ));

        final Timeline<Number> inputTimeline3 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 3),
                new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
            ));

        totalizer.inputNumberTimelines().specifyValue(
            Arrays.asList(inputTimeline1, inputTimeline2,
                inputTimeline3));

        // vérifier soigneusement la valeur exacte de la chronologie obtenue
        CREOLETestHelper.assertEquals(
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 6),
                new Interval<Number>(Date.fromISO8601("20010101"), 6.1),
                new Interval<Number>(Date.fromISO8601("20020101"), 6.3),
                new Interval<Number>(Date.fromISO8601("20030101"), 6.6)
            )),
            totalizer.totalTimeline().getValue());
    }
}

```

Figure 27. Exemple de vérification stricte d'une chronologie obtenue


```

package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
    curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class TestLaxTimelineChecking extends TestCase {

    public void testNumberSumTimeline() {

        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        final Totalizer totalizer =
            Totalizer_Factory.getFactory().newInstance(session);

        // utiliser des valeurs d'entrée qui varient au fil du temps
        final Timeline<Number> inputTimeline1 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 1),
                new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
            ));

        final Timeline<Number> inputTimeline2 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 2),
                new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
            ));

        final Timeline<Number> inputTimeline3 =
            new Timeline<Number>(Arrays.asList(
                new Interval<Number>(null, 3),
                new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
            ));

        totalizer.inputNumberTimelines().specifyValue(
            Arrays.asList(inputTimeline1, inputTimeline2,
                inputTimeline3));

        /*
         * Ne pas vérifier de manière stricte que la chronologie obtenue est exactement
         * conforme à vos prévisions, mais vérifier la valeur de la chronologie obtenue
         * à des dates particulières.
         * Il est possible que la chronologie contienne des valeurs incorrectes à
         * d'autres dates, mais selon le but de votre test, vous
         * pouvez privilégier la lisibilité à la précision des données.
         */

        final Timeline<? extends Number> resultantTimeline =
            totalizer.totalTimeline().getValue();
        CREOLETestHelper.assertEquals(6.1,
            resultantTimeline.valueOn(Date.fromISO8601("20010101")));
        CREOLETestHelper.assertEquals(6.6,
            resultantTimeline.valueOn(Date.fromISO8601("20130101")));
    }
}

```

4.5.6 Propriétés chronologiques

Chaque chronologie CER contient des propriétés importantes que vous devez connaître avant d'utiliser des chronologies dans vos jeux de règles, tests de règle CER et tout code client de CER :

- chaque chronologie CER est non modifiable (comme tous les types de données utilisés dans CER) ;
- chaque référence à une chronologie est paramétrée avec le type de valeur contenu dans la chronologie. Il peut s'agir de valeurs primitives telles que Chaîne, Date, Nombre, Booléen, etc., ou d'un type arbitrairement complexe tel qu'une classe de règles ou un autre type paramétré comme une Liste. Comme pour les autres types paramétrés dans CER, le paramètre lui-même doit être un objet non modifiable ;
- chaque chronologie CER s'étend à l'infini dans le passé et dans le futur ¹⁰. En d'autres termes, chaque chronologie CER a une valeur à *n'importe quelle* date, quelle que soit sa localisation dans le passé ou dans l'avenir ;
- lorsqu'une chronologie CER est créée, celle-ci est divisée en un ensemble d'*intervalles*, qui comportent chacun une valeur constante liée à une période comprise dans la chronologie. Les intervalles contigus présentent *toujours* des valeurs différentes ¹¹. Dans le cas contraire, elles sont fusionnées en un seul intervalle. Chaque chronologie CER s'étend indéfiniment loin dans le passé et le futur ;

Il existe un certain nombre de conséquences liées à ces propriétés :

- il n'est pas possible de laisser un vide au milieu d'une chronologie : tous les intervalles d'une chronologie doivent être contigus ;
- il n'est pas possible de démarrer une chronologie à une date particulière ; dans certains cas, vous devez choisir une valeur par défaut pertinente. Par exemple, si vous créez une Timeline<Number> de revenus issus d'un emploi, ces revenus doivent être égaux à 0 à toutes les dates précédant la date de début de l'emploi ;
- il n'est pas possible de terminer une chronologie par une date particulière : la dernière valeur de la chronologie s'applique jusqu'à nouvel ordre, c'est-à-dire arbitrairement loin dans le futur ; dans certaines circonstances, vous devez choisir une valeur par défaut pertinente. Par exemple, si vous créez une Timeline<Number> de revenus issus d'un emploi, si la date de fin de cet emploi est connue, les revenus doivent être égaux à 0 à toutes les dates postérieures à la fin de l'emploi ; dans le cas contraire, s'il n'existe aucune date de fin connue pour cet emploi, les derniers revenus doivent s'appliquer jusqu'à nouvel ordre ;
- toute tentative de création d'une chronologie qui ne comporte de valeur pour aucune date échoue. En particulier, chaque chronologie doit avoir une valeur s'appliquant à partir du *début de la durée*, représentée par une date de début null ;
- chaque chronologie peut contenir un nombre limité de changements de valeurs. Cela constitue une limite pour les chronologies représentant des valeurs qui changent un nombre de fois arbitraire. Par exemple, vous pouvez créer une Timeline<Number> représentant l'âge d'une personne, contenant la valeur 0 jusqu'à son premier anniversaire, la valeur 1 jusqu'à son deuxième anniversaire, et ainsi de suite. Pour les personnes encore en vie, il n'est pas possible de prédire le nombre d'anniversaires à venir ; c'est pourquoi une limite pratique (par exemple, 200) doit être imposée. Dans la pratique, cette limitation ne doit pas présenter de difficultés.

10. Notez que chaque chronologie couvre une durée illimitée, mais ne peut contenir qu'un nombre limité de dates dont la valeur change

11. Les valeurs identiques/différentes sont détectées par la sémantique Java `Object.equals(...)`. Tous les types utilisés en tant que type paramétré de chronologie doivent avoir des implémentations pertinentes des valeurs `Object.equals(...)` et `Object.hashCode()`.

4.5.6.1 Exemple d'intervalles de chronologie

Dans l'exemple de la situation de Joe, Mary et James, nous avons vu que Mary n'était pas chef de famille monoparentale avec mineur à charge avant de se marier avec Joe, et que lorsqu'elle s'est mariée avec Joe, elle n'était toujours pas chef de famille monoparentale avec mineur à charge, mais pour des raisons différentes.

Cela suppose que lorsque la valeur `isLoneParentOfMinorTimeline` de Mary est calculée, les chronologies d'entrée utilisées sont la `isMarriedTimeline` de Mary et son `hasMinorDependentsTimeline`.

CER identifie chaque date à laquelle les chronologies d'entrée changent, et calcule pour chacune de ces dates la valeur à obtenir (à cette date) pour déterminer si Mary est chef de famille monoparentale avec mineur à charge à cette date, comme suit :

- Dates à laquelle la `isMarriedTimeline` de Mary change :
 - 1er janvier 2001 ; et
 - 1er mai 2004.
- Dates à laquelle la `hasMinorDependentsTimeline` de Mary change :
 - 1er janvier 2001 ; et
 - 1er juin 2006.
- Par conséquent, les dates auxquelles une ou plusieurs entrées ont changé sont les suivantes :
 - 1er janvier 2001 (les deux chronologies d'entrée de Mary changent à cette date) ; et
 - 1er mai 2004 (seule la `isMarriedTimeline` de Mary change à cette date) ; et
 - 1er juin 2006 (seule la `hasMinorDependentsTimeline` de Mary change à cette date).

Ainsi, pour chacune de ces dates, vous pouvez calculer la valeur requise pour `isLoneParentOfMinorTimeline`, à l'aide de la logique de table primitive booléenne/de vérité :

Tableau 2. Calcul des valeurs d'intervalle de la `isLoneParentOfMinorTimeline` de Mary

Date à laquelle une ou plusieurs chronologies d'entrée changent de valeur	Valeur de <code>isMarriedTimeline</code> à cette date	Valeur de <code>hasMinorDependentsTimeline</code> à cette date	Valeur obligatoire de <code>isLoneParentOfMinorTimeline</code> à cette date
début de la durée (cette date est toujours incluse)	FALSE	FALSE	FALSE
1er janvier 2001	TRUE	TRUE	FALSE
1er mai 2004	FALSE	TRUE	TRUE
1er juin 2006	FALSE	FALSE	FALSE

Enfin, une chronologie est construite avec les valeurs requises pour `isLoneParentOfMinorTimeline` : à ce stade, la construction de la chronologie reconnaît que la valeur de début de durée (FALSE) et le 1er janvier 2001 (FALSE) sont identiques, et ces intervalles sont fusionnés en un seul, qui s'étend du début de la durée jusqu'au 1er mai 2004 non inclus (lorsque la valeur devient TRUE).

Remarque : La chronologie obtenue comporte des changements de valeur uniquement le 1er mai 2004 et le 1er juin 2006.

La chronologie ne contient intentionnellement *aucun* enregistrement selon lequel le 1er janvier 2001 était utilisé pendant sa construction, car la valeur de la chronologie n'a pas changé à cette date : celle-ci ne présente aucune importance pour la chronologie obtenue.

4.6 Déclenchement de recalcul en cas de changement de données

La capacité de CER à effectuer directement des recalculs est désormais remplacée par le gestionnaire de dépendance (voir Chapitre 5, «Gestionnaire de dépendance», à la page 77).

Il est recommandé de privilégier l'utilisation du gestionnaire de dépendance aux stratégies de recalcul de CER.

Chapitre 5. Gestionnaire de dépendance

L'application inclut un gestionnaire de dépendance responsable du stockage et de la gestion des dépendances entre les éléments de données d'entrée ("éléments précédents") et les éléments de données de sortie ("éléments dépendants").

CER et ses clients (comme le moteur d'éligibilité et d'autorisation ainsi que l'assistant) s'intègrent étroitement au gestionnaire de dépendance pour prendre en charge le recalcul des résultats CER dès que les entrées qui ont été utilisées dans les calculs CER changent.

Ce chapitre fournit une présentation du gestionnaire de dépendance incluant ce qui suit :

- les concepts sous-jacents du gestionnaire de dépendance et la terminologie utilisée pour les décrire ;
- les fonctions effectuées par le gestionnaire de dépendance ;
- le traitement par lots compris avec le gestionnaire de dépendance ;
- la manière dont CER s'intègre au gestionnaire de dépendance et
- la conformité du gestionnaire de dépendance.

5.1 Concepts du gestionnaire de dépendance

Lorsque la valeur d'un élément de donnée est dérivée des valeurs d'un ou de plusieurs autres éléments de données, alors la valeur dérivée *dépend* des valeurs utilisées pour l'obtenir. Si une ou plusieurs des valeurs dépendent du changement suivant, alors les éléments de données dérivés doivent être recalculés pour obtenir leur nouvelle valeur.

Le gestionnaire de dépendance utilise les termes suivants pour contenir ces concepts :

- **Élément dépendant**
Élément de donnée dérivé dont la valeur est calculée à partir d'autres éléments de données (éléments précédents).
- **Élément précédent**
Élément de donnée dont la valeur peut permettre de calculer les éléments de données dérivés (éléments dépendants).
- **Dépendance**
Enregistrement du fait que la valeur d'un élément dépendant particulier dépend de la valeur d'un élément précédent particulier.
- **Élément de changement précédent**
Enregistrement du fait que la valeur d'un élément précédent particulier a changé d'une certaine manière.
- **Ensemble de changements précédents**
Ensemble d'éléments de changements précédents regroupés pour le traitement ; utilisé pour identifier les éléments dépendants potentiellement affectés qui nécessitent un recalcul.
- **Recalcul d'élément dépendant**
Recalcul d'un élément dépendant potentiellement affecté par un ou plusieurs des changements apportés aux éléments précédents dans un ensemble de changements précédents.

Ces concepts sont explicités par un exemple.

Admettons qu'une autorisation relative à une prestation d'un demandeur est calculée à partir de données telles que :

- les informations personnelles du demandeur ;
- les informations collectées regroupées pour le dossier du demandeur et
- les taux de prestation et les seuils de revenu.

Joe, un demandeur, possède deux dossiers (123 et 124) et Mary, une autre demandeuse, possède un dossier (125). Il existe également des dossiers et des informations personnelles pour d'autres demandeurs, ainsi que des taux d'allocation utilisés pour d'autres calculs.

Dans cet exemple, l'autorisation calculée pour chaque dossier est un élément *dépendant* et les informations personnelles, les informations collectées et les taux/seuils sont des éléments *précédents*.

Il est possible de dessiner une *matrice fragmentée* qui présente les dépendances entre les éléments dépendants et les éléments précédents ("X" indiquant la présence d'une dépendance) :

Tableau 3. Exemple de matrice de dépendance

Élément précédent	Autorisation du dossier 123	Autorisation du dossier 124	Autorisation du dossier 125	Autorisation du dossier 126
Informations personnelles de Joe	X	X		
Informations personnelles de Mary			X	
Informations personnelles de Frank				
Informations collectées du dossier 123	X			
Informations collectées du dossier 124		X		
Informations collectées du dossier 125			X	
Informations collectées du dossier 126				X
Taux de prestation	X	X	X	X
Seuils de revenu	X	X	X	X
Taux d'allocation				

Observons certains exemples issus de la matrice de dépendance :

- L'autorisation du dossier 123 dépend des informations personnelles de Joe mais non de celles de Mary ;
- Les informations personnelles de Joe sont utilisées dans le calcul de ses deux dossiers (123 et 124) et
- Tous les dossiers utilisent les taux de prestation et les seuils de revenu mais pas les taux d'allocation.
- Aucune autorisation de dossier ne dépend des informations personnelles de Frank.

La matrice peut être lue comme suit :

- par colonne, pour comprendre tous les éléments précédents dont dépend un élément dépendant particulier (il s'agit de l'ensemble des dépendances qui doivent être conservées chaque fois qu'un élément dépendant est calculé) et/ou

- par ligne, pour comprendre tous les éléments dépendants dont dépend un élément précédent particulier (il s'agit de l'ensemble d'éléments dépendants qui doivent être recalculés chaque fois que la valeur d'un élément précédent change).

Au fur et à mesure que les éléments précédents et dépendants dans le système augmentent, la matrice de dépendance s'agrandit significativement. La matrice étant renseignée de façon fragmentée uniquement (c'est-à-dire que chaque élément dépendant dépend d'une petite fraction seulement des éléments précédents), les données contenues dans la matrice sont stockées uniquement pour les dépendances présentes, comme suit :

Tableau 4. Exemple de stockage de dépendance

Élément dépendant		Élément précédent
Autorisation du dossier 123	dépend de	Informations personnelles de Joe
Autorisation du dossier 123	dépend de	Informations collectées du dossier 123
Autorisation du dossier 123	dépend de	Taux de prestation
Autorisation du dossier 123	dépend de	Seuils de revenu
Autorisation du dossier 124	dépend de	Informations personnelles de Joe
Autorisation du dossier 124	dépend de	Informations collectées du dossier 124
Autorisation du dossier 124	dépend de	Taux de prestation
Autorisation du dossier 124	dépend de	Seuils de revenu
Autorisation du dossier 125	dépend de	Informations personnelles de Mary
Autorisation du dossier 125	dépend de	Informations collectées du dossier 125
Autorisation du dossier 125	dépend de	Taux de prestation
Autorisation du dossier 125	dépend de	Seuils de revenu
Autorisation du dossier 126	dépend de	Informations collectées du dossier 126
Autorisation du dossier 126	dépend de	Taux de prestation
Autorisation du dossier 126	dépend de	Seuils de revenu

(Remarque : Le tableau ci-dessus est trié par élément dépendant, ce qui permet de visualiser facilement l'ensemble des dépendances pour chaque élément dépendant, mais il peut également être trié par élément précédent pour visualiser facilement les éléments dépendants potentiellement affectés par un changement de valeur de cet élément précédent.)

Supposons que les informations personnelles de Joe soient modifiées. Etant donné que les dépendances sont enregistrées par rapport aux informations personnelles de Joe, le gestionnaire de dépendance est capable de déterminer que les dossiers 123 et 124 nécessitent un recalcul. Lorsque les dossiers sont recalculés, leurs valeurs d'autorisation changent (en raison du changement apporté aux informations personnelles de Joe). Cependant, il convient de noter que dans des situations typiques, les dépendances elles-mêmes ne changent pas. Ainsi, avant le recalcul, le dossier 123 dépendait des informations personnelles de Joe, des informations collectées stockées par rapport au dossier, des taux de prestation et des seuils de revenu et ceci reste vrai après le recalcul.

Il est possible que plusieurs valeurs d'éléments précédents changent simultanément ; par exemple, si l'organisme décide de modifier à la fois ses taux de prestation et ses seuils de revenu, alors tous les dossiers doivent être recalculés. Idéalement, chaque dossier devrait être identifié deux fois (une fois par rapport au changement des taux de prestation puis à nouveau par rapport au changement des seuils de

revenu) ; toutefois, le gestionnaire de dépendance prend en charge le groupement de ces deux changements précédents dans un seul ensemble de changements précédents. Lorsque le gestionnaire de dépendance traite l'ensemble de changements précédents, il filtre automatiquement tous les éléments dépendants en doublon de manière à ce que le travail minimal nécessaire soit effectué pour recalculer les éléments dépendants.

5.2 Fonctions du gestionnaire de dépendance

Le gestionnaire de dépendance effectue les fonctions principales suivantes :

- il stocke les enregistrements de dépendance identifiés par un client (par exemple par le moteur d'éligibilité et d'autorisation) lors du calcul d'une décision d'évaluation initiale ;
- il capture les changements des valeurs des éléments précédents qui peuvent affecter les valeurs des éléments dépendants ;
- il identifie les éléments dépendants potentiellement affectés par les éléments d'un ensemble de changements précédents ; et
- il contrôle le recalcul de ces éléments dépendants identifiés.

Ces fonctions sont décrites de manière plus détaillée dans les sections suivantes.

5.2.1 Stockage des enregistrements de dépendance

Le gestionnaire de dépendance est responsable de la création de nouveaux enregistrements de dépendance sur la base de données et de la suppression des enregistrements de dépendance existants qui ne sont plus obligatoires¹².

Chaque fois qu'un client du gestionnaire de dépendance calcule la valeur d'un élément dépendant, le client est chargé d'identifier les éléments précédents utilisés dans ce calcul, ainsi que de transmettre cet élément dépendant et ses éléments précédents au gestionnaire de dépendance. Le gestionnaire de dépendance utilise cet élément dépendant pour récupérer son ensemble de dépendances stockées existant (le cas échéant) depuis la base de données et crée ou supprime des enregistrements de dépendance alignés sur le nouvel ensemble d'éléments précédents identifiés par le client.

Généralement, la première fois que le gestionnaire de dépendance est appelé pour un élément dépendant, le gestionnaire de dépendance crée plusieurs nouvelles lignes sur la base de données pour stocker les dépendances relatives aux éléments précédents identifiés.

Toutefois, lors des appels suivants du gestionnaire de dépendance concernant le même élément dépendant, il est très fréquent que le gestionnaire de dépendance détecte que le nouvel ensemble de dépendances obligatoires transmis correspond exactement à celui déjà stocké sur la base de données. Dans ce cas, le gestionnaire de dépendance n'a pas d'écritures de base de données à effectuer. Parfois, le gestionnaire de dépendance détermine qu'un petit nombre de nouvelles lignes de dépendances sont obligatoires et/ou qu'un petit nombre de lignes de dépendances existantes sont désormais superflues et doivent être supprimées. Dans ce cas, le gestionnaire de dépendance réalise un petit nombre d'écritures de base de données afin de mettre à jour les lignes stockées par rapport aux dépendances obligatoires, en laissant la majeure partie des enregistrements de dépendance inchangés pour l'élément dépendant concerné.

Les clients du gestionnaire de dépendance peuvent identifier que des enregistrements de dépendance ne sont plus obligatoires pour un élément dépendant et ordonner au gestionnaire de dépendance de supprimer tous les enregistrements de dépendance liés à cet élément dépendant.

12. Chaque enregistrement de dépendance ne contient aucune information modifiable et le gestionnaire de dépendance ne *modifie* jamais d'enregistrements de dépendance existants - il crée seulement de nouveaux enregistrements ou supprime des enregistrements existants.

5.2.1.1 Exemple

Lorsqu'une autorisation relative à un dossier est appliquée pour la première fois, le gestionnaire de dépendance stocke les nouveaux enregistrements de dépendance pour indiquer que l'autorisation du dossier dépend des informations personnelles du demandeur, des informations collectées enregistrées concernant le dossier, des taux, etc.

Si le dossier est recalculé ultérieurement (automatiquement par le gestionnaire de dépendance suite à un changement des informations personnelles, par exemple, ou demandé manuellement par un utilisateur), après le recalcul, le gestionnaire de dépendance compare les dépendances identifiées lors du calcul avec celles déjà présentes dans la base de données et trouve des différences.

Si un nouveau membre du foyer est ajouté au dossier, lorsque l'autorisation est recalculée, une nouvelle dépendance est identifiée. Ainsi, l'autorisation du dossier dépend désormais également des informations personnelles du nouveau membre du foyer, qui s'ajoutent aux dépendances existantes déjà stockées par rapport au dossier. Le gestionnaire de dépendance crée un nouvel enregistrement de dépendance sur la base de données pour stocker la dépendance supplémentaire.

Si le nouveau membre du foyer est supprimé ultérieurement, lorsque l'autorisation est recalculée, il n'y a aucune dépendance relative aux informations personnelles du membre du foyer supprimé. Le gestionnaire de dépendance identifie que la dépendance stockée par rapport aux informations personnelles de ce membre du foyer est maintenant superflue et la supprime de la base de données, tout en conservant les autres enregistrements de dépendance (par rapport aux informations personnelles du demandeur, aux informations collectées enregistrées concernant le dossier, aux taux, etc.) intactes.

Lorsque le dossier est finalement fermé, il n'a plus besoin de la prise en charge des recalculs¹³ et les enregistrements de dépendance ne sont donc plus obligatoires. Pour un nettoyage efficace, le gestionnaire de dépendance doit supprimer tous les enregistrements de dépendance de l'autorisation du dossier. Si le dossier est rouvert ensuite, son autorisation peut être recalculée et le gestionnaire de dépendance recrée tous les enregistrements de dépendance obligatoires.

5.2.1.2 Incompréhension des éléments dépendants ou précédents

Le gestionnaire de dépendance ne conserve volontairement *pas* d'enregistrements de tous les éléments dépendants ou précédents connus dans le système, car cela aurait pour effet de :

- dupliquer toutes les données conservées ailleurs dans le système et
- générer un goulot d'étranglement lors du traitement, ce qui peut provoquer des problèmes d'accès concurrent lorsque le système détecte que de nouvelles données sont utilisées en tant qu'éléments précédents dans les calculs.

En revanche, le gestionnaire de dépendance enregistre uniquement les informations sur les dépendances. Chaque dépendance représente simplement un lien entre un élément dépendant particulier et un élément précédent particulier. Si un élément dépendant particulier n'a pas d'éléments précédents, ou inversement, aucun enregistrement de dépendance ne sera stocké pour celui-ci.

Le gestionnaire de dépendance ne comprend pas les informations relatives aux éléments dépendants et précédents stockées dans un enregistrement de dépendance. Par contre, chaque type d'élément dépendant et chaque type d'élément précédent possèdent un gestionnaire enregistré avec le gestionnaire de dépendance et ce dernier fait appel à ces gestionnaires pour effectuer un traitement métier spécifique en fonction du type, par exemple pour décoder un élément précédent ou dépendant dans une description lisible ou pour recalculer un élément dépendant si besoin.

13. Par exemple avec une stratégie de type "Do not reassess closed cases".

Voir le guide Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

5.2.1.3 Stockage de dépendance facultatif

Il est important de noter que l'utilisation du gestionnaire de dépendance pour stocker les enregistrements de dépendance est facultative.

Les clients du gestionnaire de dépendance peuvent décider si les enregistrements de dépendance sont obligatoires ou non (autrement dit, si le client a besoin ou pas de la fonction du gestionnaire de dépendance permettant d'identifier automatiquement et de recalculer les éléments dépendants).

Par exemple, le moteur d'éligibilité et d'autorisation utilise le gestionnaire de dépendance pour stocker les enregistrements de dépendance relatifs aux décisions d'évaluation de dossier (c'est-à-dire les décisions qui entraînent normalement des paiements financiers et/ou des factures). Le moteur d'éligibilité et d'autorisation nécessite que le gestionnaire de dépendance l'avertisse lorsqu'un dossier doit être réévalué, c'est pourquoi les enregistrements de dépendance doivent être stockés.

En revanche, le moteur d'éligibilité et d'autorisation contient également une fonction permettant au responsable du dossier de vérifier manuellement l'éligibilité et l'autorisation d'un dossier, d'après les informations collectées en cours d'édition. Ces calculs d'éligibilité/autorisation manuels utilisent les mêmes méthodes de calcul mais ne nécessitent aucun stockage de dépendance. En effet, le système n'a jamais à recalculer de telles décisions ; par contre, les calculs d'éligibilité/autorisation manuels sont toujours déclenchés par une demande explicite de la part d'un responsable du dossier.

5.2.1.4 Granularité des dépendances

Remarque : Les éléments de données précédents dans l'exemple ci-dessus sont délibérément vagues. Ainsi, les termes "informations personnelles" couvrent un grand nombre de zones de données personnelles comme les dates de naissance/décès, les données démographiques, etc. Le gestionnaire de dépendance ne connaît pas ou ne tient pas compte des significations des dépendances qu'il stocke entre les éléments dépendants et précédents. Il appartient aux clients du gestionnaire de dépendance d'associer des significations à celles-ci et de stocker les dépendances selon une granularité appropriée.

Le choix de la granularité implique de trouver un compromis acceptable entre les deux extrêmes suivants :

- **Granularité très fine**

Les dépendances très fines sont stockées entre les éléments dépendants et les zones de données personnelles, ce qui permet une identification extrêmement précise des éléments dépendants affectés par les changements précédents, mais au coût de très nombreux enregistrements de dépendance stockés

- **Granularité très grossière**

Des dépendances très larges sont stockées entre des éléments dépendants et les regroupements de très nombreuses zones de données personnelles dans un "élément de donnée", ce qui entraîne le stockage d'enregistrements de dépendance peu nombreux, mais risque d'impliquer la demande de faux recalculs (c'est-à-dire des recalculs qui ne sont pas nécessaires car le calcul n'est pas affecté par la zone de données personnelles spécifique qui a été modifiée).

Il est de la responsabilité des concepteurs des clients du gestionnaire de dépendance d'envisager ces compromis et de prendre des décisions raisonnables en ce qui concerne le niveau auquel stocker les informations de dépendance dans le gestionnaire de dépendance.

Par exemple, supposons que le système enregistre ces informations personnelles sur un demandeur (dans un système réaliste, il peut y avoir plusieurs zones considérées comme "informations personnelles") :

- date de naissance (utilisée dans les calculs d'autorisation) ;
- nombre d'enfants (utilisée dans les calculs d'autorisation) ; et
- nom de jeune fille de la mère (réponse à la question de sécurité, utilisée uniquement pour confirmer l'identité du demandeur ; non utilisée dans les calculs d'autorisation).

Un ensemble de dépendances à granularité très fine indique qu'une autorisation de dossier dépend de la date de naissance et du nombre d'enfants, mais pas du nom de jeune fille de la mère (étant donné qu'il n'a pas été utilisé lors des calculs) :

Tableau 5. Exemple de matrice de dépendance à granularité fine

Élément précédent	Autorisation du dossier 127
Date de naissance de Frank	X
Nombre d'enfants de Frank	X
Nom de jeune fille de la mère de Frank	

Ce stockage de dépendance à granularité fine peut finir par nécessiter le stockage de nombreuses lignes. Il convient cependant de noter que seuls les changements apportés à la date de naissance et/ou au nombre d'enfants déclenchent un recalcul de l'autorisation du dossier (si une faute de frappe est corrigée sur le nom de jeune fille de la mère uniquement, aucun recalcul d'autorisation n'est déclenché).

A l'inverse, un ensemble de dépendances à granularité très grossière représente un enregistrement beaucoup plus simple indiquant que l'autorisation du dossier dépend des informations personnelles globales :

Tableau 6. Exemple de matrice de dépendance à granularité grossière

Élément précédent	Autorisation du dossier 127
Informations personnelles de Frank	X

Ce stockage de dépendance à granularité grossière stocke moins d'enregistrements de dépendance mais si une faute de frappe est corrigée sur le nom de jeune fille de la mère, les informations personnelles globales sont modifiées et un recalcul de l'autorisation du dossier est déclenché, même si le recalcul indique que le résultat du calcul n'a pas changé.

5.2.2 Capture des éléments de changements précédents

Les clients doivent avertir le gestionnaire de dépendance dès que la valeur d'un changement précédent a été modifiée. Le gestionnaire de dépendance réunit ces changements dans un ensemble de changements précédents pour un traitement ultérieur.

L'exemple le plus courant est le moteur d'éligibilité et d'autorisation, qui contient les propagateurs d'objet de règle. Ces propagateurs sont chargés de suivre les changements apportés aux entités et aux preuves et d'indiquer ces changements au gestionnaire de dépendance.

Le gestionnaire de dépendance prend en charge les modes suivants pour la gestion des changements précédents :

- File d'attente de traitement différé (mode par défaut)
- File d'attente de traitement par lots (à l'usage des clients identifiant des changements précédents susceptibles d'entraîner le recalcul de très nombreux éléments dépendants).

Ces modes sont décrits de manière plus détaillée dans les sections suivantes.

5.2.2.1 File d'attente de traitement différé

Il s'agit du mode par défaut pour le traitement des changements précédents.

Si des éléments de changements précédents sont identifiés dans le cadre d'une transaction de base de données, le gestionnaire de dépendance :

- crée un nouvel ensemble de changements précédents unique sur la base de données ;

- ajoute tous les éléments de changements précédents identifiés au cours de la transaction à ce nouvel ensemble de changements précédents et
- met en file d'attente une demande de traitement de ce nouvel ensemble de changements précédents par un traitement différé.

5.2.2.2 File d'attente de traitement par lots

Il s'agit d'un mode spécial, utilisé uniquement par les clients qui identifient les changements apportés aux valeurs d'éléments précédents susceptibles d'entraîner le recalcul d'un grand nombre d'éléments dépendants. Le gestionnaire de dépendance conserve un ensemble de changements précédents de lots au niveau du système qui accumule les changements précédents provenant (éventuellement) de plusieurs transactions différentes.

Si des éléments de changements précédents sont identifiés dans le cadre d'une transaction de base de données, le gestionnaire de dépendance :

- récupère l'ensemble de changements précédents de lots au niveau du système ;
- ajoute tous les éléments de changements précédents identifiés au cours de la transaction à cet ensemble de changements précédents de lots et
- écrit un message d'information dans les journaux d'application pour inviter l'administrateur à planifier le traitement par lots en ce qui concerne cet ensemble de changements précédents de lots.

5.2.2.3 Exemple

Lorsque les informations personnelles de Joe sont mises à jour sur le système, le propagateur d'objet de règle d'entité avertit le gestionnaire de dépendance du changement et le gestionnaire de dépendance écrit le changement précédent sur un nouvel ensemble de changements précédents et met en file d'attente un traitement différé. Le traitement différé identifie que les deux dossiers de Joe sont potentiellement affectés et les réévalue.

Plus tard, l'administrateur publie certains changements apportés aux jeux de règles CER. Le gestionnaire de dépendance enregistre ces changements apportés aux jeux de règles CER en ajoutant un enregistrement d'élément de changement précédent à l'ensemble de changements précédents de lots au niveau du système. L'administrateur publie également certains changements apportés aux taux, et le gestionnaire de dépendance enregistre ces changements des taux en ajoutant un autre enregistrement d'élément de changement précédent à l'ensemble de changements précédents de lots au niveau du système. L'administrateur prévoit l'exécution de la suite de lots du gestionnaire de dépendance pour identifier et réévaluer les dossiers affectés.

5.2.2.4 Cycle de vie de l'ensemble de changements précédents

Chaque ensemble de changements précédents est soumis à un cycle de vie simple se présentant comme suit :

- **Ouvert**
Etat d'un ensemble de changements précédents lors de sa création initiale. Dans cet état, de nouveaux éléments de changements précédents peuvent être ajoutés à l'ensemble de changements précédents.
- **Soumis**
L'ensemble de changements précédents a été soumis dans le cadre d'un traitement lié à l'identification. Il n'est plus possible d'ajouter d'autres éléments de changements précédents à l'ensemble de changements précédents.
- **Terminé**
Le recalcul de tous les éléments dépendants affectés par les changements précédents est terminé. L'ensemble de changements précédents est conservé à des fins d'archivage uniquement.

Les transitions entre chaque état s'effectuent différemment en fonction du mode selon lequel les changements précédents ont été capturés :

- File d'attente de traitement différé :

- la transaction qui capture les changements précédents *ouvre* un nouvel ensemble de changements précédents et *soumet* celui-ci en demandant un traitement différé ; et
- le traitement différé accepte l'ensemble de changements précédents *soumis*, identifie et recalcule les éléments dépendants affectés, puis *achève* l'ensemble de changements précédents.
- File d'attente de traitement par lots :
 - la transaction qui capture les changements précédents les écrit sur l'ensemble de changements précédents de lots actuellement *ouvert* ;
 - la suite de traitements par lots du gestionnaire de dépendance effectue les étapes suivantes :
 - récupère l'ensemble de changements précédents de lots actuellement *ouvert* et *soumet* celui-ci lors de l'étape suivante du traitement par lots, puis crée un nouvel ensemble de changements précédents de lots *ouvert* pour capturer tous les autres changements précédents identifiés¹⁴;
 - effectue un traitement par lots sous forme de flux afin d'identifier et de recalculer les éléments dépendants affectés par les changements apportés à l'ensemble de changements précédents de lots *soumis* ; et
 - *achève* l'ensemble de changements précédents de lots.

5.2.3 Identification des éléments dépendants potentiellement affectés

Une fois que le gestionnaire de dépendance a capturé un ou plusieurs éléments de changement précédent et les a regroupés dans un ensemble de changements précédents, il peut identifier les éléments dépendants potentiellement affectés par un ou plusieurs des éléments de changements précédents, en examinant les enregistrements de dépendance stockés pour chaque élément précédent dans l'ensemble de changements précédents. C'est à ce stade que le gestionnaire de dépendance filtre les éléments dépendants identifiés plus d'une fois.

Cette identification des éléments dépendants potentiellement affectés se produit lors du traitement différé ou du traitement par lots, en fonction du mode activé lorsque les éléments de changements précédents ont été capturés (voir 5.2.2, «Capture des éléments de changements précédents», à la page 83).

Par exemple, si une transaction de base de données unique écrit des changements sur plusieurs lignes de base de données, alors l'étape de traitement différé identifie chaque dossier affecté une fois seulement, même si chaque ligne de base de données modifiée en elle-même affecte un dossier particulier.

De la même manière, si l'ensemble de changements précédents de lots contient des changements relatifs aux jeux de règles CER et aux taux, alors l'étape de traitement par lots identifie à nouveau chaque dossier affecté une fois seulement, même si le changement des jeux de règles CER en lui-même affecte un dossier particulier, comme le changement de taux en lui-même.

5.2.4 Recalcul des éléments dépendants identifiés

Une fois que le gestionnaire de dépendance a identifié tous les éléments dépendants susceptibles d'être affectés par les changements des éléments précédents, il demande que chacun de ces éléments dépendants soit recalculé. Le gestionnaire de dépendance ne comprend pas ce que chaque élément dépendant représente, c'est pourquoi le recalcul approprié est obtenu par le gestionnaire de dépendance en examinant un gestionnaire enregistré pour chaque élément dépendant et en déléguant la responsabilité du recalcul au gestionnaire.

Par exemple, le gestionnaire d'élément dépendant enregistré pour les décisions d'évaluations de dossiers comprend que le recalcul approprié pour un dossier donné consiste à réévaluer ce dossier.

14. Dans un système en cours d'exécution, il s'agit de la manière dont un nouvel ensemble de changements précédents de lots est créé (à savoir grâce à la soumission de son prédécesseur).

Notez cependant que l'ensemble de changements précédents de lots ouvert *initial* est fourni par un fichier DMX inclus dans l'application.

Ce recalcul d'un élément dépendant se produit lors du traitement différé ou du traitement par lots¹⁵, en fonction du mode activé lorsque les éléments de changements précédents ont été capturés (voir 5.2.2, «Capture des éléments de changements précédents», à la page 83). Une fois cette étape terminée, le système est désormais à jour par rapport aux changements précédents qui ont été capturés.

5.3 Traitement différé du gestionnaire de dépendance

La section précédente (5.2, «Fonctions du gestionnaire de dépendance», à la page 80) décrivait la manière dont le gestionnaire de dépendance prend en charge la capture des éléments de changement précédents dans un ensemble de changements précédents de lots au niveau du système, ainsi que la manière dont il contient le traitement différé pour identifier les éléments dépendants potentiellement affectés et les recalculer.

Cette section fournit davantage de détails sur le traitement différé du gestionnaire de dépendance.

Le traitement différé commence par effectuer un calcul préalable du montant total des éléments dépendants potentiellement affectés, puis du nombre de recalculs qu'il doit effectuer et prend l'une des deux décisions suivantes :

- Si ce calcul se situe au-dessous de la limite du système, les recalculs de l'ensemble de changements précédents sont alors effectués par le traitement différé ou
- Si ce calcul dépasse la limite du système, les calculs de l'ensemble de changements précédents sont déplacés vers le traitement par lots du gestionnaire de dépendance (5.4, «Traitement par lots du gestionnaire de dépendance», à la page 87). A cette étape, l'ensemble de changements précédents est marqué par un statut 'Deferred To Batch'.

5.3.1 Définition de la limite du système pour le traitement différé

La limite du système pour le traitement différé peut être définie via la propriété d'application `curam.dependency.deferred.processing.limit`.

Alors que la valeur par défaut pour cette limite est définie sur 50, il convient de noter que le vrai chiffre pour un système donné dépend de nombreux facteurs, notamment la mémoire disponible. Par conséquent, la valeur définie doit être basée sur le test du système pour trouver une valeur convenable capable de traiter la plupart, sinon la totalité des traitements différés normalement, tout en déplaçant les traitements problématiques vers le traitement par lots.

5.3.2 Traitement des erreurs dans le traitement différé

En cas d'erreur survenant au cours du traitement différé du gestionnaire de dépendance, après le nombre normal de nouveaux essais, le système déplace le travail vers le traitement par lots. Cela protège des incidents liés à des délais d'attente de transaction et garantit que toutes les possibilités de traitement des calculs ont été tentées.

En outre, le statut de l'ensemble de changements précédents dans ce dossier est modifié en 'Deferred To Batch' pour indiquer que le traitement différé a rencontré une erreur et une notification est générée pour l'émetteur du traitement différé.

15. Etant donné que chaque gestionnaire d'élément dépendant peut être appelé à partir de transactions en ligne, différées et/ou de lots, il est impératif que celui-ci n'émette *pas* d'hypothèses sur le type de transaction appliqué.

5.4 Traitement par lots du gestionnaire de dépendance

Une section précédente (5.2, «Fonctions du gestionnaire de dépendance», à la page 80) décrivait la manière dont le gestionnaire de dépendance prend en charge la capture des éléments de changements précédents dans un ensemble de changements précédents de lots au niveau du système, ainsi que la manière dont le gestionnaire de dépendance contient le traitement par lots pour identifier les éléments dépendants potentiellement affectés et les recalculer.

Cette section fournit plus de détails sur le traitement par lots du gestionnaire de dépendance.

Le gestionnaire de dépendance conserve les enregistrements de contrôle sur la base de données pour indiquer les ensembles de changements précédents de lots suivants :

- l'ensemble de changements précédents de lots actuellement ouvert (un ensemble de changements précédents de lots précisément est toujours ouvert pour accepter les nouveaux éléments de changements précédents) et
- l'ensemble de changements précédents de lots en cours de traitement par la suite de lots du gestionnaire de dépendance (le cas échéant, renseigné uniquement au cours du traitement par lots ; la plupart du temps il n'y a pas d'ensemble de changements précédents de lots dans cet état).

Ces enregistrements de contrôle sont essentiels au comportement de la suite de lots du gestionnaire de dépendance.

Lorsque des changements précédents sont en mode de file d'attente pour le traitement par lots, les journaux d'application affichent un message indiquant à l'administrateur qu'une exécution de la suite de lots du gestionnaire de dépendance est obligatoire¹⁶.

La suite de lots du gestionnaire de dépendance est constituée des traitements par lots distincts suivants :

- **Soumission de l'ensemble de changements précédents**
Point de départ de la suite de lots. Processus de flux unique simple qui soumet l'ensemble de changements précédents de lots actuellement ouvert.
- **Réalisation de nouveaux calculs de lots à partir de l'ensemble de changements précédents**
Processus de flux multiple complexe qui identifie les éléments dépendants susceptibles d'être affectés par les changements figurant dans l'ensemble de changements précédents soumis, puis les recalcule. Le temps consacré à l'exécution de ce processus varie selon le nombre de recalculs d'éléments dépendants qui sont obligatoires et peut être considérable.
- **Achèvement de l'ensemble de changements précédents**
Point final de la suite de lots. Processus de flux unique simple qui achève l'ensemble de changements précédents de lots actuellement soumis.

Ces traitements par lots sont décrits de manière plus détaillée dans les sections suivantes.

5.4.1 Soumission de l'ensemble de changements précédents

Ce traitement par lots constitue le point de départ de la suite de lots ; il contient un processus de flux unique simple qui soumet l'ensemble de changements d'éléments précédents de lots actuellement ouvert puis crée un nouvel ensemble de changements d'éléments précédents ouvert, qui sera utilisé pour capturer les éventuels changements d'éléments précédents à venir identifiés et mis en file d'attente pour le traitement par lots.

Pour exécuter ce traitement par lots, exécutez la commande suivante (sur une ligne) :

16. L'utilisateur qui a apporté les changements mis en file d'attente pour le traitement par lots reçoit également un message d'information à l'écran l'avertissant qu'une exécution de la suite de lots du gestionnaire de dépendance est obligatoire.

build runbatch -Dbatch.program=

curam.dependency.intf.SubmitPrecedentChangeSet.process

-Dbatch.username=SYSTEM

Si ce traitement par lots aboutit, il émet un message confirmant que l'ensemble de changements d'éléments précédents de lots ouvert a été soumis.

Conseil : Un erreur courante consiste à tenter d'exécuter ce traitement par lots alors qu'un autre ensemble de changements d'éléments précédents de lots est toujours à l'état soumis.

Ce traitement émet un message d'erreur simple s'il existe un autre ensemble de changements d'éléments précédents de lots qui n'a pas encore été traité entièrement par la suite de lots.

Pour plus de praticité, ce traitement par lots génère également une liste des types d'éléments dépendants enregistrés avec le gestionnaire de dépendance. Cette liste des types d'éléments dépendants est importante lors de l'exécution de la prochaine étape (5.4.2, «Effectuer des recalculs de lots à partir de l'ensemble de changements précédents»).

Les types d'éléments dépendants inclus avec l'application sont les suivants :

- **Résultat de la décision de l'évaluation de dossier**
Calcul d'une décision d'évaluation pour un dossier de livraison de produit.
Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.
- **Contexte du conseil**
Calcul du conseil.
Voir *Cúram Advisor Configuration Guide*.
- **Valeur d'attribut stockée**
Calcul d'un attribut stocké sur les tables de base de données de CER.
Voir 5.5.2, «CER utilise le gestionnaire de dépendance pour stocker les dépendances liées à des valeurs d'attribut enregistrées sur CER», à la page 97.

5.4.2 Effectuer des recalculs de lots à partir de l'ensemble de changements précédents

Ce traitement par lots correspond au processus de flux multiple complexe qui identifie les éléments dépendants susceptibles d'être affectés par les changements figurant dans l'ensemble de changements précédents soumis, puis les recalcule. Le temps consacré à l'exécution de ce processus varie selon le nombre de recalculs d'éléments dépendants qui sont obligatoires et peut être considérable.

L'étape consistant à effectuer des recalculs de lots à partir de l'ensemble de changements précédents doit être exécutée plusieurs fois : une fois pour chaque type d'élément dépendant enregistré avec le gestionnaire de dépendance (voir le résultat produit par l'étape précédente 5.4.1, «Soumission de l'ensemble de changements précédents», à la page 87). Vous êtes libre de choisir l'ordre le plus approprié selon lequel traiter les types d'éléments dépendants. Par exemple, il peut s'avérer plus essentiel pour votre entreprise de réévaluer les décisions de dossier (voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*) que d'identifier un conseil obsolète (voir le manuel *Advisor Configuration Guide*). Vous pouvez également propager ce traitement pour différents types d'éléments dépendants sur plusieurs jours, mais sachez que les éléments de changements précédents supplémentaires mis en file d'attente pour le traitement par lots ne peuvent pas être traités jusqu'à ce que l'ensemble de changements précédents actuellement soumis ait achevé la suite de lots complète du gestionnaire de dépendance.

L'étape consistant à effectuer des recalculs de lots à partir de l'ensemble de changements précédents utilise l'architecture de diffusion en flux de lots de Cúram (voir le guide Cúram Batch Performance Mechanisms) et par conséquent le traitement est divisé selon les phases suivantes :

- **Identification de blocs**

Phase, à exécuter en tant que processus unique, qui identifie les éléments dépendants (d'un type donné) potentiellement affectés par les changements figurant dans l'ensemble de changements précédents de lot soumis. Les identificateurs des éléments dépendants identifiés sont écrits dans des blocs en vue du traitement par la phase suivante.

- **Traitement des blocs**

Phase, favorisant l'exécution simultanée par plusieurs processus, qui sélectionne un bloc d'éléments dépendants identifiés et recalcule chaque élément dépendant.

Pour exécuter ce traitement par lots pour un type d'éléments dépendants particulier, exécutez la commande suivante (sur une ligne) :

```
build runbatch -Dbatch.program=  
curam.dependency.interfaces.PerformBatchRecalculationsFromPrecedentChangeSet.process  
-Dbatch.username=SYSTEM  
-Dbatch.parameters="dependentType= code-for-dependent-type "
```

Par défaut, le processus unique effectue les deux phases. Cependant, vous pouvez exécuter des processus de flux supplémentaires simultanément sur d'autres machines afin d'effectuer la deuxième phase en parallèle (voir le guide Cúram Batch Performance Mechanisms pour plus d'informations sur le traitement parallèle et les variables d'environnement qui dirigent le comportement du traitement en parallèle de ce processus consistant à effectuer des recalculs de lots à partir de l'ensemble de changements précédents). Pour exécuter un processus de flux pour un type d'élément dépendant particulier, exécutez la commande suivante (sur une ligne) :

```
build runbatch -Dbatch.program=  
curam.dependency.interfaces.PerformBatchRecalculationsFromPrecedentChangeSetStream.process  
-Dbatch.username=SYSTEM  
-Dbatch.parameters="dependentType= code-for-dependent-type "
```

Le traitement par lots ne parvient pas à démarrer avec une erreur fatale si l'une des situations suivantes se produit :

- le type d'élément dépendant n'est pas fourni ou est fourni mais ne correspond au code d'aucun type d'élément dépendant enregistré avec le gestionnaire de dépendance ou
- il n'y a pas d'ensemble de changements précédents de lots à l'état "soumis" (le processus de soumission de l'ensemble de changements précédents n'a pas encore été exécuté depuis la dernière exécution d'achèvement de l'ensemble de changements précédents).

Sinon, le traitement par lots démarre et tente d'identifier et de recalculer les éléments dépendants affectés. Le résultat de la tentative de recalcul d'un élément dépendant particulier est l'un des suivants :

- **Réussite**

L'élément dépendant a été trouvé et recalculé correctement et le traitement continue normalement.

- **Introuvable**

L'élément dépendant est introuvable et ne peut donc pas être traité. Cette situation peut se produire si un client du gestionnaire de dépendance décide qu'un élément dépendant ne doit plus exister mais oublie de demander au gestionnaire de dépendance de supprimer les enregistrements de dépendance relatifs à cet élément dépendant. Dans ces circonstances, le gestionnaire de dépendance supprime automatiquement les enregistrements de dépendance superflus et écrit un avertissement dans le résultat du flux de journaux/lots.

- **Erreur**

Une exception a été émise au cours du recalcul de l'élément dépendant (par exemple, si un calcul CER a rencontré un problème de type "division par zéro"). L'exception émise est écrite dans le résultat de flux de lots et le traitement de reprise est géré par le traitement d'évitement de l'architecture de diffusion en flux de lots de Cúram.

Lorsque ce processus consistant à effectuer des recalculs de lots à partir de l'ensemble de changements précédents est terminé, un rapport exhaustif est écrit avec des détails sur le nombre d'éléments dépendants ayant été traités avec succès par rapport aux éléments introuvables ou ayant rencontré des erreurs. Si des erreurs se produisent, vous devez examiner les journaux sortants provenant de vos flux de lots pour obtenir des informations détaillées sur les erreurs.

Conseil : Si vous définissez le niveau de journalisation Cúram standard sur "prolix¹⁷" ou un niveau supérieur, alors, avant de recalculer chaque élément dépendant, le gestionnaire de dépendance génère :

- une description lisible de l'élément dépendant et
- le sous-ensemble de changements précédents (à partir de l'ensemble de changements précédents) qui a permis l'identification de l'élément dépendant.

Ce résultat peut aider à comprendre pourquoi un élément dépendant particulier a été identifié comme nécessitant un recalcul.

Remarque : Si vous exécutez par erreur ce traitement par lots plusieurs fois pour le même type d'élément dépendant, les recalculs des éléments dépendants s'effectuent mais le recalcul identifie que l'élément dépendant est déjà à jour.

Par conséquent, ce genre d'exécution supplémentaire accidentelle pour un type d'élément dépendant n'endommage pas le système mais peut utiliser un temps important de traitement supplémentaire.

Notez attentivement la liste des types d'éléments dépendants et conservez un suivi des types d'éléments dépendants que vous avez traités et de ceux qui restent à traiter.

5.4.3 Achèvement de l'ensemble de changements précédents

Point final de la suite de lots, contenant un processus de flux unique simple qui complète l'ensemble de changements précédents de lots en cours de soumission.

ATTENTION :

N'exécutez pas ce traitement par lots avant d'être sûr que l'étape précédente (5.4.2, «Effectuer des recalculs de lots à partir de l'ensemble de changements précédents», à la page 88) est terminée pour chaque type d'élément dépendant enregistré avec le gestionnaire de dépendance.

Pour exécuter ce traitement par lots, exécutez la commande suivante (sur une ligne) :

```
build runbatch -Dbatch.program=
```

```
curam.dependency.intf.CompletePrecedentChangeSet.process
```

17. Ce niveau de journalisation est recommandé pour les environnements de développement uniquement.

La journalisation prolix peut affecter négativement les performances et l'évolutivité dans un système de production.

`-Dbatch.username=SYSTEM`

Si ce traitement par lots aboutit, il émet un message simple confirmant que l'ensemble de changements précédents de lots soumis est terminé.

Conseil : Une erreur courante consiste à tenter d'exécuter ce traitement par lots avant l'exécution de l'ensemble de changements précédents soumis.

Ce traitement émet un message d'erreur simple s'il n'y a pas d'ensemble de changements précédents de lots à l'état "soumis".

Une fois que l'ensemble de changements précédents de lots est terminé, le traitement vérifie que les éventuels nouveaux changements précédents ont été mis en file d'attente pour un traitement par lots ultérieur à compter du début de la suite de lots. Cette situation se produit si

- le recalcul d'un élément dépendant lors de l'exécution par lots est à l'origine de changements des données qui sont également utilisés comme éléments précédents ; et/ou
- le système en ligne a été exécuté simultanément à la suite de lots et un utilisateur a apporté des changements qui ont entraîné la mise en file d'attente des éléments de changements précédents pour le traitement par lots.

Le traitement émet un message simple indiquant que :

- il n'y a plus d'éléments de changements précédents mis en file d'attente pour le traitement par lots (et le système est donc à jour par rapport aux éléments de changements précédents de lots) ; ou
- des éléments de changements précédents supplémentaires ont été mis en file d'attente pour le traitement par lots, et une autre exécution de la suite de lots du gestionnaire de dépendance est obligatoire pour traiter ces éléments supplémentaires. Dans cette situation, vous devrez décider d'exécuter l'exécution supplémentaire immédiatement ou d'attendre un moment ultérieur (par exemple lorsque davantage d'éléments de changements précédents de lots ont été mis en file d'attente pour le traitement par lots).

5.5 Intégration entre CER et le gestionnaire de dépendance

CER s'intègre au gestionnaire de dépendance des différentes manières suivantes :

- CER peut identifier les dépendances pour les clients du gestionnaire de dépendance à stocker ;
- CER utilise le gestionnaire de dépendance pour stocker les dépendances des valeurs d'attributs calculés stockées sur les tables de base de données de CER ; et
- le gestionnaire de dépendance demande à CER de recalculer les valeurs d'attributs calculés stockées sur les tables de base de données de CER en cas de changement d'un élément précédent.

Ces points d'intégration entre CER et le gestionnaire de dépendance sont expliqués de manière plus détaillée dans les sections suivantes.

5.5.1 Utilitaire CER permettant d'identifier les dépendances à stocker

Un client de CER utilise CER pour réaliser des calculs complexes. Souvent, le client de CER peut également avoir besoin de stocker les dépendances dans le gestionnaire de dépendance afin que ce dernier puisse avertir le client en cas de changement d'éléments précédents et que le client puisse ensuite appeler de nouveau CER pour recalculer sa sortie, en tenant compte des changements apportés aux données précédentes.

CER contient un utilitaire permettant à ses clients d'identifier les dépendances à stocker dans le gestionnaire de dépendance. L'utilitaire prend une valeur d'attribut (calculée par CER) et retourne un ensemble d'éléments précédents pour cette valeur d'attribut. Un client de CER peut ensuite transmettre ses éléments dépendants ainsi que les éléments précédents identifiés au gestionnaire de dépendance pour stocker les enregistrements de dépendance.

Lorsque CER calcule une valeur d'attribut, il garde en mémoire une arborescence complète de dépendances logiques contenant :

- à sa racine, la valeur d'attribut calculée elle-même ;
- au niveau de ses noeuds de branche, les valeurs calculées intermédiaires (généralement sur les objets de règles internes) ;
- au niveau de ses noeuds terminaux, les données d'entrée externes récupérées lors du calcul de CER.

L'utilitaire peut analyser cette arborescence de dépendances logiques afin de fournir un ensemble d'éléments précédents très réduit, généralement basé sur les noeuds terminaux de l'arborescence. Autrement dit, les résultats du calcul intermédiaires sont ignorés et les dépendances stockées indiquent que la valeur d'attribut calculée dépend finalement des données d'entrée externes récupérées lors des calculs.

Remarque : Tout calcul complexe, tel que celui normalement effectué par CER, possède un certain nombre de valeurs dérivées intermédiaires entre les éléments dépendants globaux et ses éléments précédents d'entrée.

Ces valeurs intermédiaires ne sont pas transmises au gestionnaire de dépendance. Par contre, ce dernier stocke les enregistrements de dépendance liés aux éléments dépendants de niveau élevé (comme une autorisation de dossier) directement sur ses éléments précédents de niveau bas (comme des données d'entité, de preuves et de taux).

Les valeurs intermédiaires ne sont pas utiles lors du stockage de dépendances.

Les éléments précédents identifiés par l'utilitaire sont un mélange de ce qui suit :

- éléments précédents identifiés directement par CER
- éléments précédents identifiés par les convertisseurs d'objet de règle enregistrés avec le stockage de données de base de données de CER.

5.5.1.1 Eléments précédents identifiés directement par CER

L'utilitaire identifie directement ces types de dépendances pour une valeur d'attribut calculé :

Tableau 7. Eléments précédents identifiés directement par CER

Nom	Lorsqu'il est identifié	Déclencheur de recalcul
Valeur d'attribut enregistrée	<p>Identifie une valeur d'attribut d'entrée stockée sur les tables de base de données de CER qui ont été récupérées au cours (C'est-à-dire le calcul de l'attribut lui-même, ou sur une des valeurs d'attribut internes dont il dépend finalement (calculs intermédiaires entre la valeur d'attribut calculé et ses entrées de données externes).)du calcul global de la valeur d'attribut.</p> <p>L'identificateur d'élément précédent fait référence à l'identificateur interne de la ligne de la base de données de l'attribut stocké sur les tables de base de données de CER.</p>	Si la valeur de l'attribut stocké change, un élément de changement précédent pour la valeur de l'attribut stocké est écrit dans un ensemble de changements précédents.

Tableau 7. *Éléments précédents identifiés directement par CER (suite)*

Nom	Lorsqu'il est identifié	Déclencheur de recalcul
Définitions de jeux de règles	Identifie chaque jeu de règles contenant l'un des attributs utilisés dans le calcul global de la valeur d'attribut. L'identificateur d'élément précédent fait référence au nom du jeu de règles contenant une ou plusieurs définitions d'attributs rencontrés au cours du calcul global.	Si les changements apportés à un jeu de règles CER sont publiés, alors un élément de changement précédent relatif au jeu de règles modifié est écrit sur un ensemble de changements précédents.
recherche 'readall'	Identifie les expressions A.6.17.53, «readall», à la page 215 (sans correspondance imbriquée) rencontrées lors du calcul global, qui ont récupéré des objets de règles stockés sur les tables de base de données de CER (contrairement aux objets de règles récupérés à l'aide des convertisseurs d'objets de règles ; voir 5.5.1.2, «Éléments précédents identifiés par les convertisseurs d'objet de règle», à la page 97). L'identificateur d'élément précédent fait référence au nom de la classe de règles recherché par l'expression A.6.17.53, «readall», à la page 215.	Un élément de changement précédent pour la classe de règles est écrit sur un ensemble de changements précédents dans les cas suivants : <ul style="list-style-type: none"> • un nouvel objet de règle relatif à cette classe de règles est stocké sur les tables de base de données de CER et/ou • un objet de règle existant pour cette classe de règles est supprimé des tables de base de données de CER.
recherche 'readall/match'	Identifie les expressions A.6.17.53, «readall», à la page 215 rencontrées lors du calcul global, qui ont récupéré des objets de règles stockés sur les tables de base de données de CER (contrairement aux objets de règles récupérés à l'aide des convertisseurs d'objets de règles ; voir 5.5.1.2, «Éléments précédents identifiés par les convertisseurs d'objet de règle», à la page 97). L'identificateur d'élément précédent fait référence au nom de la classe de règles recherché par l'expression A.6.17.53, «readall», à la page 215, accompagné de critères de recherche tels que le nom et la valeur de l'attribut.	Un élément de changement précédent pour la classe de règles ainsi que son nom d'attribut et sa valeur de correspondance sont écrits sur un ensemble de changements précédents dans les cas suivants : <ul style="list-style-type: none"> • un nouvel objet de règle relatif à cette classe de règles est stocké sur les tables de base de données de CER ; • un objet de règle existant pour cette classe de règles est supprimé des tables de base de données de CER et/ou • la valeur de l'attribut utilisée dans le critère de recherche est remplacée par un objet de règle existant (auquel cas deux éléments de changements précédents sont écrits, l'un pour l'ancienne valeur de l'attribut et l'autre pour la nouvelle valeur de l'attribut).

(C'est-à-dire le calcul de l'attribut lui-même, ou sur une des valeurs d'attribut internes dont il dépend finalement (calculs intermédiaires entre la valeur d'attribut calculé et ses entrées de données externes).)		
---	--	--

Ces types de dépendances sont explicités par un exemple.

Supposons qu'un nouveau système est écrit et qu'il utilise CER pour calculer les impôts à payer d'une personne. Ce système d'impôts à payer utilise le gestionnaire de dépendance pour stocker les dépendances, de telle sorte que les impôts à payer peuvent être recalculés (à l'aide de CER) si la situation de la personne change.

Le système d'impôts à payer stocke des informations de seuils d'impôts au niveau du système dans les objets de règles, avec ces objets de règles stockés sur les tables de base de données de CER. Les règles CER concernant le calcul des impôts à payer d'une personne comprennent une expression A.6.17.53, «readall», à la page 215 pour récupérer tous les seuils d'impôts dans le système.

Le système d'impôts à payer stocke également des informations sur les actifs au niveau du système dans les objets de règles, avec ces derniers stockés sur les tables de base de données de CER. Chaque actif indique son propriétaire et sa valeur marchande. Les règles CER concernant le calcul des impôts à payer d'une personne comprennent une expression A.6.17.53, «readall», à la page 215 pour récupérer tous les actifs appartenant à cette personne (à savoir ceux présentant un identificateur Asset.ownedByPersonID correspondant à Person.personID). Il est possible de modifier la valeur marchande d'un actif et/ou de transférer un actif d'une personne à une autre en transmettant son identificateur Asset.ownedByPersonID à partir de l'identificateur d'une personne à celui d'une autre personne.

Les règles CER concernant le calcul des impôts à payer d'une personne impliquent l'addition des valeurs Asset.marketValue de tous les actifs appartenant à cette personne.

Le système d'impôts à payer contient des jeux de règles CER distincts permettant de récupérer les données d'entrée obligatoires pour le calcul des impôts à payer, contrairement aux calculs de l'entreprise actuels qui utilisent les données récupérées pour calculer les impôts à payer d'une personne.

Un utilisateur utilise le système d'impôts à payer pour calculer les impôts à payer de Joe (ID de personne 456) et de Mary (ID de personne 457), qui possèdent chacun un actif. Le système d'impôts à payer utilise l'utilitaire CER pour identifier les dépendances et les transmet au gestionnaire de dépendance pour le stockage, ce qui entraîne le stockage des dépendances suivantes :

Tableau 8. Exemple de dépendances stockées pour les impôts à payer

Type d'élément dépendant	Identificateur d'élément dépendant	Type d'élément précédent	Identificateur d'élément précédent
Impôts à payer	456 (ID de personne de Joe)	recherche 'readall'	Classe de règles : TaxThreshold (Stockée car le calcul des impôts à payer de Joe a provoqué la récupération de tous les objets de règle TaxThreshold.)
Impôts à payer	456	recherche 'readall/match'	Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=456 (Stockée car le calcul des impôts à payer de Joe a provoqué la récupération de tous les objets de règle Asset appartenant à Joe.)
Impôts à payer	456	Valeur d'attribut stockée	789 (identificateur interne de la valeur Asset.marketValue de l'actif de Joe) (Stockée car le calcul des impôts à payer de Joe a accédé à la valeur stockée marketValue sur l'actif unique récupéré pour Joe.)
Impôts à payer	456	Définitions de jeux de règles	TaxLiabilityDataRetrievalRuleSet (Stockées car le calcul des impôts à payer de Joe impliquait les définitions des attributs de règles dans TaxLiabilityDataRetrievalRuleSet (utilisé pour récupérer les objets de règles TaxThreshold et Asset).)
Impôts à payer	456	Définitions de jeux de règles	TaxLiabilityBusinessCalculationsRuleSet (Stockées car le calcul des impôts à payer de Joe impliquait les définitions des attributs de règles dans TaxLiabilityBusinessCalculationsRuleSet (utilisé pour calculer les impôts à payer globaux en fonction des données d'entrée).)

Tableau 8. Exemple de dépendances stockées pour les impôts à payer (suite)

Type d'élément dépendant	Identificateur d'élément dépendant	Type d'élément précédent	Identificateur d'élément précédent
Impôts à payer	457 (ID de personne de Mary)	recherche 'readall'	Classe de règles: TaxThreshold(Stockée car le calcul des impôts à payer de Mary a provoqué la récupération de tous les objets de règle TaxThreshold.)
Impôts à payer	457	recherche 'readall/match'	Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=457(Stockée car le calcul des impôts à payer de Mary a provoqué la récupération de tous les objets de règle Asset appartenant à Mary.)
Impôts à payer	457	Valeur d'attribut stockée	780 (identificateur interne de la valeur Asset.marketValue de l'actif de Mary)(Stockée car le calcul des impôts à payer de Mary a accédé à la valeur stockée marketValue sur l'actif unique récupéré pour Mary.)
Impôts à payer	457	Définitions de jeux de règles	TaxLiabilityDataRetrievalRuleSet(Stockées car le calcul des impôts à payer de Mary impliquait les définitions des attributs de règles dans TaxLiabilityDataRetrievalRuleSet (utilisé pour récupérer les objets de règles TaxThreshold et Asset).)
Impôts à payer	457	Définitions de jeux de règles	TaxLiabilityBusinessCalculationsRuleSet(Stockées car le calcul des impôts à payer de Mary impliquait les définitions des attributs de règles dans TaxLiabilityBusinessCalculationsRuleSet (utilisé pour calculer les impôts à payer globaux en fonction des données d'entrée).)

<p>(Stockée car le calcul des impôts à payer de Joe a provoqué la récupération de tous les objets de règle TaxThreshold.)(Stockée car le calcul des impôts à payer de Joe a provoqué la récupération de tous les objets de règle Asset appartenant à Joe.)(Stockée car le calcul des impôts à payer de Joe a accédé à la valeur stockée marketValue sur l'actif unique récupéré pour Joe.)(Stockées car le calcul des impôts à payer de Joe impliquait les définitions des attributs de règles dans TaxLiabilityDataRetrievalRuleSet (utilisé pour récupérer les objets de règle TaxThreshold et Asset.)(Stockées car le calcul des impôts à payer de Joe impliquait les définitions des attributs de règles dans TaxLiabilityBusinessCalculationsRuleSet (utilisé pour calculer les impôts à payer globaux en fonction des données d'entrée.)(Stockée car le calcul des impôts à payer de Mary a provoqué la récupération de tous les objets de règle TaxThreshold.)(Stockée car le calcul des impôts à payer de Mary a provoqué la récupération de tous les objets de règle Asset appartenant à Mary.)(Stockée car le calcul des impôts à payer de Mary a accédé à la valeur stockée marketValue sur l'actif unique récupéré pour Mary.)(Stockées car le calcul des impôts à payer de Mary impliquait les définitions des attributs de règles dans TaxLiabilityDataRetrievalRuleSet (utilisé pour récupérer les objets de règles TaxThreshold et Asset.)(Stockées car le calcul des impôts à payer de Mary impliquait les définitions des attributs de règles dans TaxLiabilityBusinessCalculationsRuleSet (utilisé pour calculer les impôts à payer globaux en fonction des données</p>	<p>TaxLiabilityDataRetrievalRuleSet</p> <p>TaxLiabilityBusinessCalculationsRuleSet</p> <p>TaxLiabilityDataRetrievalRuleSet</p> <p>TaxLiabilityBusinessCalculationsRuleSet</p>		
<p>96</p>	<p>© 2011 Cummins Financial Solutions</p>	<p>Cúram Express Rules Reference Manual</p>	

Nous pouvons voir maintenant comment les recalculs des impôts à payer sont déclenchés par divers changements de données :

Tableau 9. Exemple d'éléments de changements précédents pour les impôts à payer

Changements de données	Éléments de changements précédents enregistrés	Recalculs déclenchés
La valeur marchande de l'actif de Joe augmente de 100 \$ à 120 \$	<ul style="list-style-type: none"> Valeur d'attribut stockée, 789 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés
Mary vend son actif et son objet de règle est supprimé	<ul style="list-style-type: none"> recherche 'readall/match', Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=457 	<ul style="list-style-type: none"> Les impôts à payer de Mary sont recalculés
Joe reçoit un nouvel actif, stocké en tant que nouvel objet de règle	<ul style="list-style-type: none"> recherche 'readall/match', Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=456 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés
Joe transfère son premier actif à Mary, l'élément ownedByPersonID de l'actif passe donc de 456 à 457	<ul style="list-style-type: none"> recherche 'readall/match', Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=456 (ancienne valeur) recherche 'readall/match', Classe de règles : Asset, avec la valeur d'attribut ownedByPersonID=457 (nouvelle valeur) 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés Les impôts à payer de Mary sont recalculés
Un administrateur présente un nouveau seuil d'impôts, stocké en tant que nouvel objet de règle	<ul style="list-style-type: none"> recherche 'readall', Classe de règles : TaxThreshold 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés Les impôts à payer de Mary sont recalculés
Un administrateur supprime un seuil d'impôts existant, ce qui supprime son objet de règle	<ul style="list-style-type: none"> recherche 'readall', Classe de règles : TaxThreshold 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés Les impôts à payer de Mary sont recalculés
Un administrateur publie les changements apportés au jeu de règles TaxLiabilityBusinessCalculationsRuleSet	<ul style="list-style-type: none"> Définitions de jeux de règles, TaxLiabilityBusinessCalculationsRuleSet 	<ul style="list-style-type: none"> Les impôts à payer de Joe sont recalculés Les impôts à payer de Mary sont recalculés

5.5.1.2 Éléments précédents identifiés par les convertisseurs d'objet de règle

Les convertisseurs d'objet de règle enregistrés avec un stockage de données de la base de données de CER peuvent également contribuer aux dépendances identifiées par l'utilitaire CER.

Voir la documentation relative à chaque convertisseur d'objet de règle pour en savoir plus sur les dépendances identifiées.

5.5.2 CER utilise le gestionnaire de dépendance pour stocker les dépendances liées à des valeurs d'attribut enregistrées sur CER

Pour les objets de règles stockés sur les tables de base de données de CER, chaque valeur d'attribut sur ces objets de règles peut jouer un rôle tel que :

- dépendant - valeur d'attribut obtenue par un calcul à partir des données d'entrée
- précédent - valeur d'attribut utilisée en tant que données d'entrée lors du calcul d'un élément dépendant.

CER enregistre un gestionnaire dépendant et un gestionnaire précédent avec le gestionnaire de dépendance pour permettre que ses valeurs d'attribut enregistrées soient utilisées en tant qu'éléments dépendants et/ou précédents dans les dépendances stockées.

Par exemple, si un attribut `totalIncome` est présent sur un objet de règle `Person` stocké sur les tables de base de données de CER et si le calcul de l'attribut `totalIncome` impliqué dans la récupération des valeurs d'attribut `Income.amount` également stockées sur les tables de base de données de CER, alors CER doit demander au gestionnaire de dépendance d'enregistrer le fait que la valeur d'attribut `Person.totalIncome` dépend des valeurs d'attribut `Income.amount`.

5.5.3 Le gestionnaire de dépendance demande à CER de recalculer toutes les valeurs d'attributs calculés stockées

Lorsque les valeurs d'éléments précédents changent, le gestionnaire de dépendance identifie les valeurs d'attributs stockées dans CER qui dépendent de ces valeurs d'éléments précédents modifiées et demande à CER de recalculer ses valeurs d'éléments dépendants via le gestionnaire d'élément dépendant enregistré par CER avec le gestionnaire de dépendance.

Par exemple, si une valeur d'attribut `Income.amount` stockée sur les tables de base de données de CER a changé de valeur, alors CER avertit le gestionnaire de dépendance que l'élément précédent `Income.amount` a changé et le gestionnaire de dépendance détermine ensuite que l'élément dépendant `Person.totalIncome` nécessite un recalcul, puis demande à CER de le recalculer.

5.6 Conformité

Voir D.6, «Gestionnaire de dépendance», à la page 259 pour connaître l'instruction de conformité du gestionnaire de dépendance.

Chapitre 6. A propos de l'éditeur CER

6.1 Introduction

L'éditeur CER facilite la création et la maintenance des jeux de règles CER. Certains jeux de règles sont issus de la législation et d'autres servent à aider les utilisateurs à effectuer certaines activités. Par conséquent, l'éditeur CER possède deux vues de base : la vue métier pour les utilisateurs qui connaissent la structure et le texte requis par la législation et la vue technique pour ceux qui connaissent les aspects de l'implémentation du développement de règles.

L'éditeur CER contient une barre de menus globale composée de fonctions utilisateur courantes comme *Enregistrer*, *Rechercher* et *Annuler et Rétablir*. Il existe d'autres options *Enregistrer* pour permettre à l'utilisateur d'effectuer les actions *Exporter*, *Valider* et *Enregistrer tout* sur les jeux de règles.

6.2 Menu global de l'éditeur CER

L'éditeur CER fournit une fonctionnalité commune intégrée au menu global pour un accès simplifié.

6.2.1 Menu global de l'éditeur CER

Tableau 10. Barre de menus

Nom	Description
Annuler	Pour annuler la dernière édition, sélectionnez Annuler dans la barre de menus.
Rétablir	Pour annuler la dernière action Annuler, sélectionnez Rétablir dans la barre de menus ; cette opération annule la dernière action effectuée.
Inclure les jeux de règles	Permet d'afficher les jeux de règles et autorise l'ajout de jeux de règles en fournissant un chemin d'accès aux classes.
Exporter	L'éditeur CER prend en charge l'exportation de tous les diagrammes dans la vue de présentation des règles vers des images PNG qui peuvent être enregistrées sous forme de fichier d'archive ZIP unique sur le disque dur local de l'utilisateur.
Enregistrer	Un jeu de règles peut être enregistré à tout moment une fois qu'il est ouvert dans l'éditeur. Un jeu de règles ayant un statut de résultats publiés dans un enregistrement «en cours d'édition» créé pour ce jeu de règles est en cours d'enregistrement. Les modifications que vous apportez sont enregistrées sur l'enregistrement en cours d'édition. L'enregistrement d'un jeu de règles ayant un statut «en cours d'édition» enregistre vos modifications directement. Les jeux de règles peuvent également avoir un statut «Récemment créé». L'enregistrement des modifications sur un jeu de règles récemment créé entraîne l'enregistrement des modifications directement dans ce jeu de règles.
Enregistrer tout	L'éditeur CER prend en charge l'ouverture de plusieurs jeux de règles au sein d'une même instance de l'éditeur. Un certain nombre d'éléments de CER peuvent indiquer des classes ou des attributs qui sont définis dans les autres jeux de règles. Les éléments de CER indiquant un élément contenu dans un autre jeu de règles possèdent une option de menu <i>Ouvrir le jeu de règles</i> . Cela permet à l'utilisateur de disposer d'un nombre quelconque de jeux de règles ouverts dans l'éditeur à tout moment et simultanément. L'option de menu <i>Enregistrer tout</i> enregistre tous les jeux de règles] qui sont ouverts dans l'éditeur et qui ont été modifiés.

Tableau 10. Barre de menus (suite)

Nom	Description
Valider	Permet à l'utilisateur de valider ses changements apportés à un jeu de règles. Le valideur de jeu de règles du moteur de règles CER est appelé. Les jeux de règles CER sont des fichiers XML qui respectent le schéma de règles fourni par CER. CER inclut également un valideur de jeu de règles complet capable de détecter les erreurs dans votre jeu de règles avant d'autoriser l'exécution de ce dernier. Le valideur de jeu de règles CER fournit une liste des erreurs dans le panneau Propriétés et validations afin qu'elles puissent être résolues par l'utilisateur. En cas d'erreurs, le valideur de jeu de règles CER signale les erreurs et les arrêts de traitement.
Rechercher	Permet à l'utilisateur d'effectuer une recherche de texte rapide qui ouvre une fenêtre contextuelle de résultats de recherche contenant les résultats correspondants. Pour plus d'informations sur la fonctionnalité proposée et la méthode de recherche avancée, voir la section Outils de recherche.

6.2.2 Outils de recherche de l'éditeur CER

6.2.2.1 Critères de recherche de l'éditeur CER

Un utilisateur peut effectuer une recherche de texte générale pour les critères suivants (Règles et Références de règles, Descriptions, Dossiers, Données techniques et Tables de codes) :

Tableau 11. Critères de recherches

Nom	Description
Règles et Références de règles	Recherche d'une règle à l'aide du nom de la règle.
Descriptions	Recherche des règles ou des attributs au sein d'une règles en entrant une partie du texte utilisé pour décrire la règle ou l'attribut.
Dossiers	Recherche d'un dossier à l'aide du nom du dossier.
Données techniques	Recherche des attributs correspondant à des attributs spécifiés ou non spécifiés.
Tables de codes	Recherche des attributs spécifiés en tant que tables de codes.

6.2.3 Vue métier

La vue métier fournit au professionnel une vue centralisée des règles où les éléments appropriés à la création et la maintenance des règles métier sont disponibles. Elle se compose d'une arborescence ou d'une vue hiérarchique des règles, des grilles de règles, des palettes d'éléments métier et des propriétés ainsi que d'un panneau de validation.

6.2.3.1 Vue de présentation des règles

Cette vue affiche toutes les règles de niveau supérieur ainsi que les dossiers où elles se trouvent. Un menu contextuel est disponible sur chaque élément dans la vue de présentation :

Tableau 12. Nouveaux éléments de menu

Nom	Description
Dossier (Bouton)	Crée un dossier en fournissant un nouveau nom de dossier.
Règle (Bouton)	Crée une nouvelle règle en fournissant un nouveau nom de règle et le type de données de cette règle qui sera défini par défaut sur un type booléen. La règle sera créée dans le dossier sélectionné. Si aucun dossier n'est sélectionné, la nouvelle règle sera créée au niveau racine.
Nouveau dossier	Crée un dossier en fournissant un nouveau nom de dossier.

Tableau 12. Nouveaux éléments de menu (suite)

Nom	Description
Nouvelle règle	Crée une nouvelle règle en fournissant un nouveau nom de règle et le type de données de cette règle qui sera défini par défaut sur un type booléen. La règle sera créée dans le dossier sélectionné. Si aucun dossier n'est sélectionné, la nouvelle règle sera créée au niveau racine.
Supprimer	Supprime un dossier ou une règle en fonction des éléments mis en évidence dans la vue de présentation.
Définir le type de règle	Créer une dérivation pour un attribut. Une liste des types de données est fournie pour une nouvelle règle.
Déplacer la règle	Cela permet au développeur de règles de déplacer une règle d'un dossier vers un autre dossier en sélectionnant le nom du dossier.
Rechercher les références à cette règle	Permet à l'utilisateur de rechercher toutes les références à la règle sélectionnée.

6.2.4 Vue technique

L'onglet Technique affiche toutes les classes et attributs de la règle en cours d'édition. Vous pouvez supprimer toutes les classes et attributs en cliquant à nouveau sur le menu contextuel associé à chaque classe ou attribut. Les actions disponibles dans l'onglet Métier de la vue Structure sont les suivantes :

6.2.4.1 Vue de présentation de classe

Cette vue affiche toutes les règles de niveau supérieur ainsi que les dossiers où elles se trouvent. Un menu contextuel est disponible sur chaque élément dans la vue de présentation :

Tableau 13. Nouveaux éléments de menu

Nom	Description
Classe (Bouton)	Crée une classe en fournissant un nouveau nom de classe.
Attribut (Bouton)	Crée un nouvel attribut au sein d'une classe donnée en fournissant un nouveau nom d'attribut. Ce bouton est activé uniquement une fois qu'une classe est sélectionnée à partir de la vue de présentation, sinon il reste inactif.
Nouvel attribut	Crée un nouvel attribut au sein d'une classe donnée en fournissant un nouveau nom d'attribut.
Supprimer	Supprime une classe ou un attribut en fonction des éléments mis en évidence dans la vue d'arborescence.
Définir le type de règle	Permet au développeur de règles de créer une dérivation pour un attribut.
Rechercher les références à cette règle	Permet à l'utilisateur de rechercher toutes les références à la règle sélectionnée.

6.3 Grille du diagramme

La grille du diagramme fournit les commandes de glisser-déposer, de bascule technique ainsi que de panoramique et de zoom

6.3.1 Glisser-déposer

Il est possible de faire glisser et déposer un élément de règle à partir de n'importe quelle palette sur le diagramme. Par exemple, glissez et déposez un élément "rule" sur un attribut. Un indicateur visuel apparaît sur la cible si celle-ci peut accepter que l'on fasse glisser l'élément de règle.

De plus, les éléments de règles peuvent être glissés et déposés entre leurs conteneurs. Par exemple, faites glisser et déposer un élément "rule" provenant d'une section de résultat de l'élément de règle "when" sur une section de résultat d'un autre élément de règle "when".

6.3.2 Basculer sur Afficher les diagrammes détaillés

L'éditeur CER fournit deux types différents de vues pour les éléments de règle. La vue métier est une version simple destinée au rédacteur de règle métier. La vue technique est plus détaillée et est principalement destinée au concepteur de règle technique. L'éditeur CER propose une option permettant de basculer entre les vues technique et métier. L'icône de basculement se trouve au-dessus des commandes de panoramique et de zoom sur la grille de règles.

6.3.3 Commandes de panoramique et de zoom

L'éditeur CER fournit des commandes de panoramique et de zoom pour permettre à l'utilisateur de mieux visualiser et éditer le jeu de règles. Les flèches situées sur la commande circulaire peuvent être utilisées pour faire un panoramique des règles très étendues. Le fait de cliquer sur le bouton central de la commande de panoramique recentre l'image sur la position de démarrage. Le bouton Plus peut être utilisé pour effectuer un zoom avant et le bouton Moins peut être utilisé pour effectuer un zoom arrière :



Figure 29. Commandes de panoramique et de zoom

6.4 Palettes d'outils et de modèles

L'éditeur CER fournit quatre types de palette d'éléments de règle et trois palettes de modèles de règle. Ces palettes contiennent les éléments suivants : Métier (par défaut), Métier (étendu), Types de données et Technique. Ces modèles sont regroupés dans les catégories suivantes : Unités du ménage, Unités financières, Unités d'assistance alimentaire et Table de décision.

6.4.1 Palettes applicatives

Les palettes applicatives, par défaut et étendues, incluent une plage d'éléments de règles qui peuvent être utilisés pour concevoir la logique applicative sous la forme d'un diagramme.

Tableau 14. Eléments de règles sur les palettes applicatives

Image	Nom	Description
	Rule	L'élément Rule fournit une représentation graphique de l'expression 'reference'. Voir 7.2.1, «Rule», à la page 111.
	And Rule Group	L'élément And Rule Group fournit une représentation graphique de l'expression 'all' et renvoie une valeur booléenne. Voir 7.2.2, «And Rule Group», à la page 113.
	Or Rule Group	L'élément Or Rule Group fournit une représentation graphique de l'expression 'any' et renvoie une valeur booléenne. Voir 7.2.3, «Or Rule Group», à la page 113.
	Not	L'élément Not fournit une représentation graphique de l'expression 'not' et annule une valeur booléenne. Voir 7.2.4, «Not», à la page 114.
	Choose	L'élément Choose fournit une représentation graphique de l'expression 'choose' et choisit une valeur d'après une condition satisfaite. Voir 7.2.5, «Choose», à la page 114.

Tableau 14. Eléments de règles sur les palettes applicatives (suite)

Image	Nom	Description
	Compare	L'élément Compare fournit une représentation graphique de l'expression 'compare' et compare une valeur située à gauche avec une valeur située à droite, en fonction de la comparaison fournie. Voir 7.2.6, «Compare», à la page 114.
	When	L'élément When fait partie d'un élément Choose et contient une condition à tester, et une valeur à renvoyer si la condition est transmise. Voir 7.2.7, «When», à la page 115.
	Arithmetic	L'élément Arithmetic fournit une représentation graphique de l'expression 'arithmetic' et effectue un calcul arithmétique sur deux nombres (un nombre situé à gauche et un nombre situé à droite) et arrondit éventuellement le résultat au nombre indiqué de décimales. Voir 7.2.8, «Arithmetic», à la page 115.
	MIN	L'élément Min fournit une représentation graphique de l'expression 'min' et détermine la plus petite valeur dans une liste (ou la valeur Null si la liste est vide). Voir 7.2.9, «MIN», à la page 115.
	MAX	L'élément Max fournit une représentation graphique de l'expression 'max' et détermine la plus grande valeur dans une liste (ou la valeur Null si la liste est vide). Voir 7.2.10, «MAX», à la page 115.
	Sum	L'élément SUM fournit une représentation graphique de l'expression 'sum' et calcule la somme numérique d'une liste de valeurs Number. Voir 7.2.11, «SUM», à la page 116.
	Repeating Rule	L'élément Repeating Rule fournit une représentation graphique de l'expression 'dynamiclist' et crée une nouvelle liste en évaluant une expression sur chaque élément dans une liste existante. Voir 7.2.12, «Repeating Rule», à la page 116.
	Filter	L'élément Filter fournit une représentation graphique de l'expression 'filter' et crée une nouvelle liste contenant tous les éléments d'une liste existante qui remplissent la condition de filtre. Voir 7.2.13, «Filter», à la page 116.
	Size	L'élément Size fournit une représentation graphique de l'expression 'property' et le nom de la propriété est défini sur 'size'. Voir 7.2.14, «Size», à la page 117.
	Otherwise	L'élément Otherwise fait partie de l'élément Choose et contient une valeur à renvoyer. Voir 7.2.15, «Otherwise», à la page 117.
	Legislation Change	L'élément Legislation Change fournit une représentation graphique de l'expression 'legislationchange' et peut contenir plusieurs éléments Era. Voir 7.2.16, «Legislation Change», à la page 118.
	Era	L'élément Era fait partie des éléments Legislation Change et contient une entrée de date (date de début vide), ainsi qu'une valeur à renvoyer à l'élément Legislation Change. Voir 7.2.17, «Era», à la page 118.

6.4.2 Palette Types de données

La palette Types de données comprend une plage d'éléments de règles qui peuvent être utilisés pour concevoir les types de données sous la forme d'un diagramme.

Tableau 15. Eléments de règles sur la palette Types de données

Image	Nom	Description
	Boolean	L'élément Boolean fournit une représentation graphique des expressions 'true' et 'false'. Par défaut, l'élément Boolean est défini sur true. Voir 7.3.1, «Boolean», à la page 118.
	String	L'élément String fournit une représentation graphique de l'expression 'String' qui correspond à une valeur constante Number littérale. Voir 7.3.2, «String», à la page 118.
	Number	L'élément Number fournit une représentation graphique de l'expression 'Number' qui correspond à une valeur constante Number littérale. Voir 7.3.3, «Number», à la page 119.
	Date	L'élément Date fournit une représentation graphique de l'expression 'Date' qui correspond à une valeur constante Date littérale. Voir 7.3.4, «Date», à la page 119.
	Codetable	L'élément Codetable fournit une représentation graphique de l'expression 'Code' qui correspond à une valeur constante littérale représentant un code provenant d'une table de codes Cúram. Voir 7.3.5, «Code Table», à la page 119.
	Rate	L'élément Rate fournit une représentation graphique de l'expression 'Rate' qui correspond à une valeur constante littérale représentant un taux provenant d'une table de taux. Voir 7.3.6, «Rate», à la page 120.
	Frequency Pattern	L'élément Frequency Pattern fournit une représentation graphique de l'expression 'FrequencyPattern' qui correspond à une valeur constante FrequencyPattern littérale. Voir 7.3.7, «Frequency Pattern», à la page 120.
	Resource Message	L'élément Resource Message fournit une représentation graphique de l'expression 'ResourceMessage' et crée un message localisable à partir d'une ressource de propriété. Voir 7.3.8, «Resource Message», à la page 120.
	XML Message	L'élément XML Message fournit une représentation graphique de l'expression 'XMLMessage' et crée un message localisable à partir d'une ressource de propriété. Voir 7.3.9, «Xml Message», à la page 121.
	Null	L'élément Null fournit une représentation graphique de l'expression 'Null'. L'élément Null peut être utilisé au sein d'éléments comme Choose, When, Compare etc... pour comparer une valeur donnée à la valeur 'Null'. Voir 7.3.10, «Null», à la page 121.

6.4.3 Palette technique

La palette technique comprend une série d'éléments de règle qui peuvent être utilisés pour concevoir la logique technique sous forme de diagramme.

Tableau 16. Eléments de règle sur palette technique

Image	Nom	Description
	Create	L'élément Create fournit une représentation graphique de l'expression create et obtient une nouvelle instance d'une classe de règles dans la mémoire de session. Voir 7.4.1, «Create», à la page 122.

Tableau 16. Eléments de règle sur palette technique (suite)

Image	Nom	Description
	Search	L'élément Search fournit une représentation graphique de l'expression readall et extrait toutes les instances d'objet de règle d'une classe de règles qui ont été créées par le code client. Voir 7.4.2, «Search», à la page 122.
	Fixed List	L'élément Fixed List fournit une représentation graphique de l'expression fixedlist et crée une liste à partir des éléments connus au moment de la conception du jeu de règles. Voir 7.4.3, «Fixed List», à la page 123.
	Property	L'élément Property fournit une représentation graphique de l'expression property. L'élément Property obtient une propriété d'un objet Java. Voir 7.4.4, «Property», à la page 123.
	Custom Expression	Custom Expression fournit une représentation graphique de n'importe quel noeud XML valide défini par l'utilisateur. Voir 7.4.5, «Custom Expression», à la page 124.
	Existence Timeline	L'élément Existence Timeline fournit une représentation graphique de l'expression existencetimeline. Voir 7.4.6, «Existence Timeline», à la page 124.
	Timeline	L'élément Timeline fournit une représentation graphique de l'expression Timeline. Voir 7.4.7, «Timeline», à la page 125.
	Interval	L'élément Interval fournit une représentation graphique de l'expression Interval. Voir 7.4.8, «Interval», à la page 126.
	Combine Succession Set	L'élément Combine Succession Set fournit une représentation graphique de l'expression combineSuccessionSet. Voir 7.4.9, «CombineSuccessionSet», à la page 126.
	Call	L'élément Call fournit une représentation graphique de l'expression call et fait appel à une méthode Java statique permettant d'effectuer un calcul complexe. Voir 7.4.10, «Call», à la page 126.
	Period Length	L'élément Period Length fournit une représentation graphique de l'expression periodlength et calcule la quantité d'unités de temps entre deux dates. Voir 7.4.11, «Period Length», à la page 127.
	ALL	L'élément ALL fournit une représentation graphique de l'expression all et renvoie une valeur booléenne. Voir 7.4.12, «ALL», à la page 128.
	ANY	L'élément ANY fournit une représentation graphique de l'expression any et renvoie une valeur booléenne. Voir 7.4.13, «ANY», à la page 128.
	This	L'élément This fournit une représentation graphique de l'expression this qui est une référence à l'objet de règle en cours. Voir 7.4.14, «This», à la page 128.
	Sort	L'élément Sort fournit une représentation graphique de l'expression sort. Il prend une liste en tant qu'élément enfant et effectue un tri sur celle-ci. Voir 7.4.15, «Sort», à la page 128.
	Shared Rule Reference	Shared Rule Reference est en fait une référence contenant un élément Create. Voir 7.4.16, «Shared Rule Reference», à la page 129.
	CONCAT	L'élément CONCAT fournit une représentation graphique de l'expression concat et crée un message en concaténant une liste de valeurs. Voir 7.4.17, «CONCAT», à la page 130.

Tableau 16. Eléments de règle sur palette technique (suite)

Image	Nom	Description
	Join Lists	L'élément Join Lists fournit une représentation graphique de l'expression joinlists et crée une liste en joignant plusieurs listes existantes. Voir 7.4.18, «Join Lists», à la page 130.

6.4.4 Modèle d'unités de foyer

Le modèle d'unités de foyer comprend une plage d'éléments de règles qui peuvent être utilisés pour concevoir les unités de foyer sous la forme d'un diagramme.

Tableau 17. Eléments de règles sur la palette Modèle d'unités de foyer

Image	Nom	Description
	Composition du foyer	Voir 7.5.1, «Composition du foyer», à la page 131.
	Catégorie de foyer	Voir 7.5.2, «Catégorie de foyer», à la page 131.

6.4.5 Modèle d'unités financières

Le modèle d'unités financières inclut une plage d'éléments de règles qui peuvent être utilisés pour concevoir les unités financières sous la forme d'un diagramme.

Tableau 18. Eléments de règles sur la palette Modèle d'unités financières

Image	Nom	Description
	Unité financière	Voir 7.6.1, «Unité financière», à la page 131.
	Catégorie d'unité financière	Voir 7.6.2, «Catégorie d'unité financière», à la page 131.
	Membre d'unité financière	Voir 7.6.3, «Membre d'unité financière», à la page 131.

6.4.6 Modèle d'unités d'assistance alimentaire

Le modèle d'unités d'assistance alimentaire comprend une plage d'éléments de règles qui peuvent être utilisés pour concevoir les unités d'assistance alimentaire sous la forme d'un diagramme.

Tableau 19. Eléments de règles sur la palette Modèle d'unités d'assistance alimentaire

Image	Nom	Description
	Unité d'assistance alimentaire	TODO : lien vers l'affichage de l'unité d'assistance alimentaire
	Catégorie d'assistance alimentaire pour une personne seule	TODO : lien vers l'affichage de la catégorie d'assistance alimentaire pour une personne seule 7.7.2, «Catégorie d'assistance alimentaire pour une personne seule», à la page 132.
	Catégorie d'assistance alimentaire pour plusieurs personnes	Voir 7.7.3, «Catégorie d'assistance alimentaire pour plusieurs personnes», à la page 132.
	Membres de groupe de repas	Voir 7.7.4, «Membres de groupe de repas», à la page 132.

Tableau 19. Eléments de règles sur la palette Modèle d'unités d'assistance alimentaire (suite)

Image	Nom	Description
	Proches	Voir 7.7.5, «Proches», à la page 132.
	Annuler	Voir 7.7.6, «Annuler», à la page 132.
	Membre d'une unité de foyer	Voir 7.7.7, «Membre du foyer», à la page 132.
	Membres facultatifs	Voir 7.7.8, «Membres facultatifs», à la page 132.
	Exceptions	Voir 7.7.9, «Exceptions», à la page 133.

6.4.7 Modèle de table de décision

Le modèle de table de décision contient l'élément de table de décision utilisé pour créer des tables de décision.

Tableau 20. Eléments sur la palette Table de décision

Image	Nom	Description
	Decision Table	L'élément Decision Table fournit une représentation graphique de l'expression 'decision table'. Voir 7.8.1, «Decision Table», à la page 133 pour plus d'informations.

6.5 Menus contextuels d'éléments de règles

Le menu contextuel d'éléments de règles fournit des éléments généraux (fonctions) pour tous les éléments de règles et les éléments spécifiques (fonctions) pour un élément de règle individuel. Voir Chapitre 7, «Référence des éléments de règle pour les palettes de l'éditeur CER», à la page 111 pour obtenir les éléments de menu contextuel d'éléments de règles spécifiques.

Tableau 21. Eléments de menus contextuels généraux

Nom	Description
Couper	Coupe l'élément de règle et prépare le déplacement vers un autre emplacement.
Copier	Copie l'élément de règle existant et prépare la création d'une nouvelle copie dans un autre emplacement.
Supprimer	Supprime l'élément de règle du diagramme.
Réduire	Réduit l'enfant de l'élément de règle.
Développer	Développe l'enfant de l'élément de règle.
Créer chronologie	Uniquement disponible pour la vue technique. Crée une chronologie pour l'élément de règle.
Créer intervalle de chronologie	Uniquement disponible pour la vue technique. Crée un intervalle de chronologie pour l'élément de règle.
Supprimer chronologie	Uniquement disponible pour la vue technique. Supprime une chronologie de l'élément de règle.
Supprimer intervalle de chronologie	Uniquement disponible pour la vue technique. Supprime un intervalle de chronologie de l'élément de règle.

6.6 Propriétés d'éléments de règles

Cette section décrit les propriétés générales, de classe de règles et d'attribut. Voir Chapitre 7, «Référence des éléments de règle pour les palettes de l'éditeur CER», à la page 111 pour connaître les propriétés d'éléments de règles spécifiques.

6.6.1 Panneaux de propriété et de validation réductibles

Les panneaux de propriété et de validation peuvent être développés et réduits en cliquant sur le bouton à bascule situé sur la bordure interne du panneau. Le développement des panneaux de propriété et de validation révèle les détails des propriétés du diagramme actuellement sélectionné. La réduction du panneau de propriété et de validation masque ces détails pour laisser plus d'espace à l'affichage de la grille du diagramme.

Les panneaux de propriété ont été regroupés par onglets métier et technique pour afficher diverses informations de propriété basées sur le point de vue du développeur de règles. Par exemple, un développeur de règles métier n'est pas censé entrer les onglets d'un élément de règle car ce niveau de détail relève de la responsabilité du développeur de règles techniques.

6.6.2 Propriétés générales de tous les éléments de règles

Tableau 22. Propriétés générales

Nom	Description
Nom d'affichage	Il s'agit d'un nom localisable. Cette zone prend en charge les caractères multi-octets et accentués. Cette zone est disponible dans l'onglet Métier.
Description	La description prend en charge les entrées de texte libre pour les règles ou les attributs. Les caractères multi-octets et accentués sont pris en charge. Le développeur de règles doit entrer une description métier significative de la règle ou de l'attribut. Cette zone est disponible dans l'onglet Métier. Le contenu de la zone de description est localisable et peut contenir des caractères multi-octets ou accentués.
Lien de législation	Lien vers un site Web de législation. Cette zone est disponible dans l'onglet Métier.
ID de nom d'affichage	ID de nom de l'élément de règle. Cette zone est disponible dans l'onglet Technique.
Balise	L'annotation de balise prend en charge les entrées de balise de texte libre pour tous les éléments qui prennent en charge les annotations. Elle est utilisée pour la recherche de l'élément de règle. Cette zone est disponible dans l'onglet Métier.

6.6.3 Propriétés de classe de règles

Tableau 23. Propriétés de classe de règles

Nom	Description
Attribut principal	Sélectionnez un attribut dans la liste déroulante en tant qu'attribut principal de cette classe de règles. Celle-ci est visible dans l'onglet Métier.
Abstrait	La classe sera une classe abstraite si cette propriété est sélectionnée. Celle-ci est visible dans l'onglet Métier.
Extensions	Les utilisateurs peuvent étendre cette classe de règles à partir d'une autre classe de règles qui appartient au jeu de règles en cours ou externe défini par le jeu de règles de changement et l'assistant de classe de règles. Celle-ci est visible dans l'onglet Métier.
Classe	Nom de la classe de règles. Celle-ci est visible dans l'onglet Technique.

Tableau 23. Propriétés de classe de règles (suite)

Nom	Description
SuccessionSet	L'annotation SuccessionSet sera ajoutée si cette propriété est sélectionnée. Les utilisateurs doivent sélectionner des attributs de date dans les listes déroulantes "Attribut de date de début" et "Attribut de date de fin". Celle-ci est visible dans l'onglet Technique.
ActiveInEditSuccessionSet	L'annotation ActiveInEditSuccessionSet sera ajoutée si cette propriété est sélectionnée. Les utilisateurs doivent sélectionner des attributs de date dans les listes déroulantes "Attribut de date de début" et "Attribut de date de fin". Celle-ci est visible dans l'onglet Technique.

6.6.4 Propriétés d'attribut

Tableau 24. Propriétés d'attribut

Nom	Description
Type de données	Type de données d'un attribut. Si l'utilisateur souhaite modifier le type de données dans une chronologie, cochez la case Chronologie. Si l'utilisateur souhaite modifier le type de données dans une liste, cochez la case Liste. Celle-ci est visible dans l'onglet Métier.
Affichage	L'annotation Affichage sera ajoutée si cette propriété est sélectionnée. Elle nécessite que les utilisateurs entrent une valeur. Celle-ci est visible dans l'onglet Métier.
Sous-écran d'affichage	L'annotation Sous-écran d'affichage sera ajoutée si cette propriété est sélectionnée. Celle-ci est visible dans l'onglet Métier.
Classe	Nom de la classe de règles à laquelle appartient l'attribut. Celle-ci est visible dans l'onglet Technique.
Attribut	Nom de l'attribut. Celle-ci est visible dans l'onglet Technique.
Ensemble de succession connexe	L'annotation Ensemble de succession connexe sera ajoutée si cette propriété est sélectionnée. Elle nécessite que les utilisateurs sélectionnent l'une des successions (aucune, parent, enfant) dans la liste déroulante. Celle-ci est visible dans l'onglet Technique.
Type d'informations collectées connexes	L'annotation Type d'informations collectées connexes sera ajoutée si cette propriété est sélectionnée. Elle nécessite que les utilisateurs sélectionnent l'une des informations collectées connexes (aucune, parent, enfant) dans la liste déroulante. Celle-ci est visible dans l'onglet Technique.
Abstrait	L'attribut sera défini sur un élément de règle abstrait et la classe sera une classe abstraite si cette propriété est sélectionnée. Celle-ci est visible dans l'onglet Technique.

6.7 Assistants d'éléments de règles

Assistant de jeu de règles de changement et de classe de règles :

Cet assistant est utilisé par certains éléments de règles (par exemple Règle ou Créer un élément de règle) pour établir un lien vers une autre classe de règles d'un jeu de règles en cours ou d'un jeu de règles externe. Les trois options disponibles dans cet assistant sont :

Tableau 25. Table de l'assistant d'élément de règle

Nom	Description
Créer une référence de règle vide	Cela permet au rédacteur de règles de créer une marque de réservation pour un élément de règle qui peut être édité ultérieurement lors du développement de la règle.
Créer une nouvelle règle	Cela permet au rédacteur de règles de créer une nouvelle règle en indiquant le nom de la règle et le type de résultat de la nouvelle règle dans la zone de liste déroulante.
Utiliser une règle existante	Cela permet au rédacteur de règles de choisir une règle déjà créée en sélectionnant la règle voulue dans la zone de liste déroulante. Si la règle ne figure pas dans les règles existantes, le rédacteur de règles peut sélectionner un autre jeu de règles en entrant le nom du jeu de règle souhaité et en appuyant sur le bouton de recherche. Cette action ouvre une autre boîte de dialogue, ce qui permet au rédacteur de règles de sélectionner le jeu de règles approprié.

Chapitre 7. Référence des éléments de règle pour les palettes de l'éditeur CER

7.1 Introduction

Cette section permet de définir tous les éléments de règles inclus dans l'éditeur CER. Les éléments de règles ci-dessous sont classés selon différents types de palettes d'éléments de règles ; consultez les sections précédentes pour obtenir des informations utiles sur le classement de ces éléments de règles.

7.1.1 Palettes

L'ancrage des palettes peut être supprimé en déplaçant la palette de son emplacement d'origine sur la grille du diagramme. Pour ancrer à nouveau la palette, faites-la simplement glisser vers la droite de l'écran.

7.1.2 Palettes réductibles

Le conteneur de palette peut être développé et réduit en cliquant sur le bouton à bascule situé au niveau de la bordure des conteneurs de palette. Lorsque vous développez le conteneur de palette, le nom de la palette actuellement sélectionnée et les descriptions de chaque élément de règle apparaissent. Lorsque vous réduisez le conteneur de palette, les descriptions sont masquées et la quantité d'espace que le conteneur reprend sur le diagramme est réduite.

7.2 Référence des éléments de règle pour les palettes métier étendues et par défaut

7.2.1 Rule

L'élément Rule fournit une représentation graphique de l'expression 'rule' et annule une valeur booléenne. Aucun autre élément de la palette ne peut être ajouté à l'élément Reference. L'élément Rule peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou Repeating Rule. Sept types de scénarios permettent d'utiliser les éléments Rule. Une référence à une règle peut être ajoutée lorsqu'un utilisateur est dans la vue Métier (voir Section 2.3.1, de *Working With CER*) et dans la vue technique (Voir Section 2.3.2, de *Working With CER*). Lorsqu'un utilisateur est dans la vue Métier, les options suivantes sont disponibles :

- Référence vide - les utilisateurs peuvent créer une référence vide ;
- Créer une nouvelle règle - les utilisateurs peuvent créer une nouvelle règle ;
- Utiliser une règle existante - les utilisateurs peuvent sélectionner une règle à partir du jeu de règles en cours, ou effectuer une recherche de jeux de règles externes et sélectionner une règle à partir d'un jeu de règles externe.

Lorsqu'un utilisateur se trouve dans la vue technique, les options suivantes sont disponibles :

- Utiliser une règle existante - les utilisateurs peuvent sélectionner une règle à partir du jeu de règles en cours, ou effectuer une recherche de jeux de règles externes et sélectionner une règle à partir d'un jeu de règles externe ;
- Créer une nouvelle règle - les utilisateurs peuvent créer une nouvelle règle.

Si un utilisateur se trouve dans la vue Métier et qu'il souhaite ajouter une référence à un attribut ou une classe de règles spécifique, celui-ci doit basculer vers la vue technique.

Tableau 26. Types de scénarios à utiliser dans les éléments Rule

Nom	Description
Référence imbriquée	Une référence imbriquée peut être créée sur la référence qui pointe vers un attribut d'un autre type de classe de règle, par exemple, elle ne se trouve pas dans la classe existante. L'attribut de référence le plus externe correspond à un objet de la classe d'attributs de référence interne. Cette structure peut être créée uniquement si l'attribut de référence interne, qui est défini sur un autre type de classe de règles, est situé dans la classe dans laquelle la référence interne est créée.
Référence imbriquée avec Créer	Une référence imbriquée peut être créée sur la référence qui pointe vers un attribut d'un autre type de classe de règle, par exemple, elle ne se trouve pas dans la classe existante, cependant l'attribut n'est pas situé dans la classe en cours. L'attribut de référence le plus externe correspond à un objet de la classe d'attributs de référence interne. Cette structure peut être créée uniquement si l'attribut de référence interne (qui est défini sur un autre type de classe de règles) est situé dans la classe qui ne correspond pas à la classe dans laquelle la référence interne est créée.
CurrentUnitMemeber	Pour faire référence au membre dans l'unité actuelle qui remplit la condition du test exceptionnel.
FARelationship	Pour faire référence à la classe utilisée comme enregistrement de relation pour le membre du groupe de repas.
FAException	Pour faire référence à une classe utilisée pour vérifier si d'autres membres d'une unité remplissent la condition exceptionnelle.
HCCurrent	Pour faire référence à un élément de liste en cours (comme le membre du ménage, etc.) dans la composition du foyer utilisée, HCCurrent est utilisé.
En cours	Pour faire référence à un élément de liste en cours des listes comme dynamiclist, l'élément En cours est utilisé. Si une référence à un attribut de la classe d'éléments de liste en cours doit être effectuée, alors une référence pointant vers cet attribut entoure l'élément en cours.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 27. Eléments de propriétés de Reference

Nom	Description
Classe	Le nom de la classe de règles. Celui-ci est visible dans l'onglet Métier.
Attribut	Le nom de l'attribut. Celui-ci est visible dans l'onglet Métier.
Élément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Cet élément est activé lorsque l'option Élément unique est sélectionnée.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return first, return last) lorsque plusieurs éléments sont trouvés. Cet élément est activé lorsque l'option Élément unique est sélectionnée.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 28. Eléments des menus contextuels de l'élément Rule

Nom	Description
Encapsuler dans OR	Encapsule l'élément Rule dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Rule dans l'élément And Rule Group.
Editer la règle	Permet d'éditer l'élément Rule en sélectionnant la règle à laquelle vous souhaitez faire référence. Consultez l'assistant d'édition de règles ci-après.
Ouvrir un diagramme pour cette règle	Ouvre le diagramme pour la règle référencée dans un nouvel onglet de diagramme.
Inclure une logique de règle ici	Si l'élément Rule fait référence à une autre règle (dans le même jeu de règles ou dans un jeu de règles externe), la logique de cette règle peut être incluse à la règle actuellement affichée dans l'éditeur.

7.2.2 And Rule Group

L'élément And Rule Group fournit une représentation graphique de l'expression 'all' et renvoie une valeur booléenne. Il opère sur une liste de valeurs booléennes pour déterminer si toutes les valeurs de la liste sont vraies. La liste des valeurs booléennes est généralement fournie par un élément fixedlist. D'autres éléments de la palette (Reference, Repeating Rule ou Choose) fournissant une liste de valeurs booléennes peuvent être ajoutés au membre vide de l'élément And pour le calcul. L'élément And Rule Group peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group or When.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 29. Eléments des menus contextuels de l'élément And Rule Group

Nom	Description
Encapsuler dans AND	Encapsule l'élément And Rule Group dans un autre élément And Rule Group.
Encapsuler dans OR	Encapsule l'élément And Rule Group dans l'élément Or Rule Group.
Remplacer par Or	Remplace l'élément And Rule Group par l'élément Or Rule Group.

7.2.3 Or Rule Group

L'élément Or Rule Group fournit une représentation graphique de l'expression 'any' de valeurs booléennes pouvant être ajoutées au membre vide de l'élément Or Rule Group pour le calcul. L'élément Or Rule Group peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou When.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 30. Eléments des menus contextuels de l'élément Or

Nom	Description
Encapsuler dans OR	Encapsule l'élément Or Rule Group dans un autre élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Or Rule Group dans l'élément And Rule Group.
Remplacer par And	Remplace l'élément Or Rule Group par l'élément And Rule Group.

7.2.4 Not

L'élément Not fournit une représentation graphique de l'expression 'not' et annule une valeur booléenne. D'autres éléments de la palette, par exemple, Reference, Or ou And renvoyant une valeur booléenne peuvent être ajoutés à l'élément Not. L'élément Not peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 31. Eléments des menus contextuels de l'élément Not

Nom	Description
Encapsuler dans And	Encapsule l'élément Not dans un autre élément And Rule Group.
Encapsuler dans Or	Encapsule l'élément Not dans l'élément or Rule Group.

7.2.5 Choose

L'élément Choose fournit une représentation graphique de l'expression 'choose' et sélectionne une valeur basée sur une condition respectée. L'assistant d'édition de la sélection fournit neuf types de données : Chaîne, Nombre, Booléen, Date, Date-heure, Table de codes, Message, Chronologie, Table de codes, Classe de règles, Classe Java et Message. Seuls les éléments When et Otherwise peuvent être ajoutés à l'élément Choose. L'élément Choose peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou la vue complexe de l'élément Compare.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 32. Eléments des menus contextuels de l'élément Choose

Nom	Description
Encapsuler dans OR	Encapsule l'élément Choose dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Choose dans l'élément And Rule Group.
Editer un choix	Fournit une liste des types de données pour Choose. Consultez la boîte de dialogue d'édition de la sélection ci-après.

7.2.6 Compare

L'élément Compare fournit une représentation graphique de l'expression 'compare' et compare une valeur de gauche à une valeur de droite, en fonction de la comparaison fournie. Lorsqu'un élément de comparaison est ajouté à un diagramme, il crée des membres vides pour les arguments gauche et droit. L'élément de comparaison peut être ajouté à d'autres éléments de la palette, par exemple, When, And Rule Group ou Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 33. Eléments des menus contextuels de l'élément Compare

Nom	Description
Encapsuler dans OR	Encapsule l'élément Compare dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Compare dans l'élément And Rule Group.

7.2.7 When

L'élément When fait partie de l'élément Choose et contient une condition à tester, et une valeur à renvoyer si la condition est validée. D'autres éléments de la palette, par exemple, Code ou Any, peuvent être ajoutés à la condition vide de l'élément When. D'autres éléments de la palette, par exemple, Number ou Reference, peuvent être ajoutés à la valeur vide de l'élément When.

7.2.8 Arithmetic

L'élément Arithmetic fournit une représentation graphique de l'expression 'arithmetic' et effectue un calcul arithmétique de deux nombres (à gauche et à droite), puis arrondit le résultat (facultatif) au nombre spécifié de décimales. D'autres éléments de la palette (Max, Min ou Reference) peuvent être ajoutés à l'élément Arithmetic. Celui-ci peut être ajouté à d'autres éléments de la palette, par exemple, WHEN ou Repeating Rule.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 34. Eléments de propriétés d'Arithmetic

Nom	Description
Décimales	Emplacement permettant à un utilisateur d'entrer le nombre de décimales. Celui-ci est visible dans l'onglet Métier.
Arrondi	Fournit différents types d'arrondis (entier supérieur, inférieur, entier inférieur, moitié inférieure, entier pair le plus proche, moitié supérieure, supérieur). Cet élément est visible dans l'onglet Métier.

7.2.9 MIN

L'élément Min fournit une représentation graphique de l'expression 'min' et détermine la valeur la moins élevée d'une liste (ou null si la liste est vide). L'élément Min comporte un élément fixedlist contenant un type d'objet comparable. D'autres éléments de la palette (Number ou Rule Reference) peuvent être ajoutés au membre vide (fixedlist) de l'élément Min. L'élément Min peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 35. Eléments des menus contextuels de l'élément Min

Nom	Description
Définir sur Max	Remplace l'élément Min par l'élément Max.

7.2.10 MAX

L'élément Max fournit une représentation graphique de l'expression 'max' et détermine la valeur la plus importante d'une liste (ou null si la liste est vide). L'élément Max comporte un élément fixedlist contenant un type d'objet comparable. D'autres éléments de la palette (Number ou Rule Reference) peuvent être ajoutés au membre vide (fixedlist) de l'élément Max. L'élément Max peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 36. Eléments des menus contextuels de l'élément Max

Nom	Description
Définir sur Min	Remplace l'élément Max par l'élément Min.

7.2.11 SUM

L'élément SUM fournit une représentation graphique de l'expression 'sum' et calcule la somme numérique d'une liste de valeurs de Number. L'élément SUM comporte un élément fixedlist contenant un type d'objet numérique quelconque. D'autres éléments de la palette, par exemple, Number ou Rule Reference, peuvent être ajoutés au membre vide (fixedlist) de l'élément SUM. L'élément SUM peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

7.2.12 Repeating Rule

L'élément Repeating Rule fournit une représentation graphique de l'expression 'dynamiclist' et crée une nouvelle liste en évaluant une expression sur chaque élément d'une liste existante. D'autres éléments de la palette, par exemple, Rule Reference, peuvent être ajoutés à la liste vide de l'élément Repeating Rule. D'autres éléments de la palette, par exemple, Sum ou Choose, peuvent être ajoutés au membre vide (listitemexpression) de l'élément Repeating Rule. L'élément Repeating Rule peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou When.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 37. Eléments de propriétés de Repeating Rule

Nom	Description
Elément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Cet élément est activé lorsque 'Elément unique' est coché.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Cet élément est activé lorsque 'Elément unique' est coché.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 38. Eléments des menus contextuels de Repeating Rule

Nom	Description
Retirer des doublons	Permet de supprimer les éléments en double dans l'élément Repeating Rule.
Concaténer des résultats	Permet de concaténer les éléments dans l'élément Repeating Rule.
Joindre des listes internes	Permet d'associer les listes pour n'en former qu'une.
Réussites sur n'importe lequel	Encapsule l'élément Repeating Rule dans l'élément Any.
Réussites sur tous	Encapsule l'élément Repeating Rule dans l'élément All.
Eléments de somme	Additionne une liste de nombres.

7.2.13 Filter

L'élément Filter fournit une représentation graphique de l'expression 'filter' et crée une nouvelle liste contenant tous les éléments d'une liste existante respectant la condition du filtre. D'autres éléments de la

palette (Reference) peuvent être ajoutés à la liste vide de l'élément Filter. D'autres éléments de la palette (Sum ou Choose) peuvent être ajoutés au membre vide (listitemexpression) de l'élément Filter. L'élément Filter peut être ajoutés à d'autres éléments de la palette, par exemple, And, Or ou When.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 39. Eléments de propriétés de Filter

Nom	Description
Elément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Cet élément est activé lorsque 'Elément unique' est coché.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return first, return last) lorsque plusieurs éléments sont trouvés. Cet élément est activé lorsque 'Elément unique' est coché.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 40. Eléments des menus contextuels de Filter

Nom	Description
Retirer des doublons	Permet de supprimer les éléments en double dans l'élément Filter.
Concaténer des résultats	Permet de concaténer les éléments dans l'élément Filter.
Joindre des listes internes	Permet d'associer les listes pour n'en former qu'une.
Encapsuler dans OR	Encapsule l'élément Filter dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Filter dans l'élément And Rule Group.

7.2.14 Size

L'élément Size fournit une représentation graphique de l'expression 'property' et le nom de cet élément est défini sur 'size'. L'élément Size obtient une propriété d'un objet Java. D'autres éléments de la palette, par exemple, Rule Reference ou Repeating Rule, peuvent être ajoutés à l'élément Size. L'élément Size peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 41. Eléments de propriétés de Size

Nom	Description
Valeur	Le nom de l'élément de propriété.

7.2.15 Otherwise

L'élément Otherwise fait partie de l'élément Choose et contient une valeur à renvoyer. D'autres éléments de la palette, par exemple, Number ou Rule Reference peuvent être ajoutés à la valeur vide de l'élément Otherwise.

7.2.16 Legislation Change

L'élément Legislation Change fournit une représentation graphique de 'legislationchange' et peut contenir plusieurs éléments Era. L'assistant de changement de législation fournit dix types de données (Chaîne, Nombre, Booléen, Date, Date-heure, Liste, Message, Table de codes, Classe de règles et Classe Java). Le type de données sélectionné est utilisé pour définir une valeur retournée de l'élément Era.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 42. Eléments des menus contextuels de l'élément Legislation Change

Nom	Description
Editer un changement de législation	Permet d'éditer le changement de législation en sélectionnant un type pour la nouvelle règle. Consultez la boîte de dialogue d'édition du changement de législation ci-après.

7.2.17 Era

L'élément Era fait partie des éléments Legislation Change et contient une entrée de date (vide à partir de), et une valeur à renvoyer à l'élément Legislation Change. D'autres éléments de la palette (Date or Rule Reference) peuvent être ajoutés à l'entrée de date vide de l'élément Era. D'autres éléments de la palette (Choose ou Reference) peuvent être ajoutés à la valeur vide de l'élément Era.

7.3 Référence des éléments de règle pour la palette de types de données

7.3.1 Boolean

L'élément Boolean fournit une représentation graphique de l'expression 'true' et 'false'. Par défaut, l'élément Boolean est défini sur true. L'élément Boolean peut être ajouté à d'autres éléments de la palette, par exemple, les éléments And Rule Group, Or Rule Group ou Repeating Rule. Aucun élément ne peut être ajouté à l'élément Boolean.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 43. Eléments des menus contextuels de Boolean

Nom	Description
Définir sur False	Définit l'élément de règle booléen sur false si sa valeur est true.
Définir sur True	Définit l'élément de règle booléen sur true si sa valeur est false.

7.3.2 String

L'élément String fournit une représentation graphique de l'expression 'String' correspondant à une valeur constante Number littérale. L'élément String peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule. Aucun élément ne peut être ajouté à l'élément String.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 44. Eléments de propriétés de String

Nom	Description
Valeur	La valeur de la chaîne. Cet élément est visible dans l'onglet Métier.

7.3.3 Number

L'élément Number fournit une représentation graphique de l'expression 'Number' correspondant à une valeur constante Number littérale. L'élément Number peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule. Aucun élément ne peut être ajouté à l'élément Number.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 45. Eléments de propriétés de Number

Nom	Description
Valeur	La valeur du nombre. Cet élément est visible dans l'onglet Métier.

7.3.4 Date

L'élément Date fournit une représentation graphique de l'expression 'Date' correspondant à une valeur constante Date littérale. L'élément Date peut être ajouté à d'autres éléments de la palette, par exemple, PeriodLength ou Era. Aucun élément ne peut être ajouté à l'élément Date.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 46. Eléments de propriétés de Date

Nom	Description
Valeur	La valeur de la date. La valeur par défaut est définie sur la date du jour. Cet élément est visible dans l'onglet Métier.
Date zéro	En cochant l'option 'Date zéro', l'utilisateur peut utiliser l'option Cúram 'Date zéro'. Cet élément est visible dans l'onglet Métier.

7.3.5 Code Table

L'élément Code Table fournit une représentation graphique de l'expression 'Code', qui correspond à une valeur constante littérale représentant un code d'une table de codes Cúram. L'élément Code Table peut être ajouté à d'autres éléments de la palette, par exemple, When. Aucun élément ne peut être ajouté à l'élément Code Table.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 47. Eléments de propriétés de Code Table

Nom	Description
Nom de table de codes	Nom de la table de codes. Laissez cette zone vide et recherchez toutes les tables de codes disponibles. Entrez une valeur et recherchez une table de codes commençant par la valeur entrée. Cet élément est visible dans l'onglet Métier.
Valeur de table de codes	Une liste déroulante contenant tous les éléments contenus dans la table de codes sélectionnée. Cet élément est visible dans l'onglet Métier.

7.3.6 Rate

L'élément Rate fournit une représentation graphique de l'expression 'rate' qui correspond à une valeur constante littérale représentant un taux d'une table de taux Cúram. L'élément Rate peut être ajouté à d'autres éléments de la palette, par exemple, When. Aucun élément ne peut être ajouté à l'élément Rate.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 48. Eléments de propriétés de Rate

Nom	Description
Nom de la table	Le nom de la table de taux. Cet élément est visible dans l'onglet Métier.
Ligne	La valeur de la ligne de la table de taux. Cet élément est visible dans l'onglet Métier.
Colonne	La valeur de la colonne de la table de taux. Cet élément est visible dans l'onglet Métier.

7.3.7 Frequency Pattern

L'élément Frequency Pattern fournit une représentation graphique de l'expression 'FrequencyPattern' correspondant à une valeur constante FrequencyPattern littérale. L'élément Frequency Pattern peut être ajouté à d'autres éléments de la palette, par exemple, When ou Create. Aucun élément ne peut être ajouté à l'élément Frequency Pattern.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 49. Eléments de propriétés de Frequency Pattern

Nom	Description
Modèle	Le modèle de fréquence. Cet élément est visible dans l'onglet Métier.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 50. Eléments des menus contextuels de Filter

Nom	Description
Editer un modèle de fréquence	Permet d'éditer le modèle de fréquence sélectionné.

7.3.8 Resource Message

L'élément Resource Message fournit une représentation graphique de l'expression 'ResourceMessage' et crée un message localisable à partir d'une propriété de ressource. L'élément Resource Message peut être ajouté à d'autres éléments de la palette, par exemple, When ou Create. Aucun élément ne peut être ajouté à l'élément Resource Message.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 51. Eléments de propriétés de Resource Message

Nom	Description
Clé	Les objets d'un regroupement de ressources contiennent un ensemble de paires clé-valeur. Vous devez spécifier la clé, qui doit correspondre à une chaîne, lorsque vous souhaitez extraire la valeur du regroupement de ressources. Cet élément est visible dans l'onglet Métier.
Regroupement de ressources	Le nom du regroupement de ressources. Cet élément est visible dans l'onglet Métier.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 52. Eléments des menus contextuels de Resource Message

Nom	Description
Nouvel argument	Permet d'ajouter un nouvel argument à l'élément de règle du message de ressource.
Supprimer l'argument	Permet de supprimer un argument de l'élément de règle du message de ressource.

7.3.9 Xml Message

L'élément Xml Message fournit une représentation graphique de l'expression 'XmlMessage' et crée un message à partir du contenu XML à structure libre. L'élément Xml Message peut être ajouté à d'autres éléments de la palette, par exemple, When ou Create. Aucun élément ne peut être ajouté à l'élément Xml Message.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 53. Eléments de propriétés de XML Message

Nom	Description
Valeur	La valeur du contenu de la ressource Xml.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 54. Eléments des menus contextuels de Xml Message

Nom	Description
Editer le message	Fournit une boîte de dialogue dans laquelle l'utilisateur peut saisir le contenu du message.

7.3.10 Null

L'élément null fournit une représentation graphique de 'null'. L'élément null peut être utilisé dans des éléments tels que Choose/When, Compare, etc. pour comparer des valeurs avec la valeur null.

7.4 Référence des éléments de règle pour la palette de logique technique

7.4.1 Create

L'élément Create fournit une représentation graphique de l'expression 'create' et obtient une nouvelle instance d'une classe de règles dans la mémoire de la session. L'élément Create prend en charge deux types de paramètres (standard et obligatoire). D'autres éléments de la palette (Rule Reference, Repeating Rule ou Choose) peuvent être ajoutés aux paramètres de l'élément Create. L'élément Create peut être ajouté à d'autres éléments de la palette, par exemple, Repeating Rule or When.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 55. Eléments de propriétés de Create

Nom	Description
Classe	Le nom de la classe de règles choisie comme type. Cet élément est visible dans l'onglet technique.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 56. Eléments des menus contextuels de l'élément Create

Nom	Description
Editer Create	Permet d'éditer l'élément Create en sélectionnant la classe et le jeu de règles auxquels vous souhaitez faire référence.
Nouveau Paramètre	Permet de créer un nouveau paramètre en sélectionnant l'attribut auquel vous souhaitez ajouter un paramètre.
Nouveau paramètre obligatoire	Permet de créer un paramètre obligatoire.

7.4.2 Search

L'élément Search fournit une représentation graphique de l'expression 'readall' et extrait toutes les instances d'objet de règle d'une classe de règles qui ont été créées par le code client. Il peut extraire un élément unique d'une liste si l'expression 'singleitem' est sélectionnée. L'élément Search peut être ajouté à d'autres éléments de la palette, par exemple, Filter ou Create. Aucun élément ne peut être ajouté à l'élément Search.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 57. Eléments de propriétés de Search

Nom	Description
Classe	Le nom de la classe de règles choisie comme type. Cet élément est visible dans l'onglet technique.
Jeu de règles	Le nom du jeu de règles incluant la classe de règles sélectionnée. Cet élément est visible dans l'onglet technique.
Élément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.

Tableau 57. Eléments de propriétés de Search (suite)

Nom	Description
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Celui-ci est activé lorsque l'option 'Elément unique' est sélectionnée.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return first, return last) lorsque plusieurs éléments sont trouvés. Celui-ci est activé lorsque l'option 'Elément unique' est sélectionnée.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 58. Eléments des menus contextuels de l'élément Search

Nom	Description
Editer la recherche	Editez l'élément de recherche en sélectionnant la classe et le jeu de règles auxquels vous souhaitez faire référence.

7.4.3 Fixed List

L'élément Fixed List fournit une représentation graphique de l'expression 'fixedlist' et crée une nouvelle liste à partir des éléments connus au moment de la création du jeu de règles. L'assistant de l'élément Fixed List fournit neuf types de données : Chaîne, Nombre, Booléen, Date, Date-heure, Entrée de table de codes, Classe de règles, Classe Java et Message. La fonction de sous-liste est fournie. D'autres éléments de la palette (Rule Reference, Repeating Rule ou Choose) peuvent être ajoutés au membre vide de l'élément Fixed List. L'élément Fixed List peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou When. L'élément Fixed List sera encapsulé par un autre élément de règle en fonction du type de données sélectionné. Par exemple, les éléments de règles And Rule Group/Or Rule Group pour le type booléen, l'élément de règle Concat pour le type chaîne et les éléments de règles Max/Min/Somme pour le type nombre.

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 59. Eléments de propriétés FixedList

Nom	Description
Type de données	Le type de données de fixedlist. Celui-ci doit correspondre au type de données de l'attribut. Si l'utilisateur souhaite remplacer le type de données par une chronologie, cochez la case correspondante. Si l'utilisateur souhaite remplacer le type de données par une liste, cochez la case correspondante. Cet élément est visible dans l'onglet Métier.

7.4.4 Property

L'élément Property fournit une représentation graphique de l'expression 'property'. L'élément Property obtient une propriété d'un objet Java. D'autres éléments de la palette, par exemple, Rule Reference ou Repeating Rule, peuvent être ajoutés à l'élément Property. L'élément Property peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 60. Eléments de propriétés de Property

Nom	Description
Valeur	Le nom de l'élément de propriété.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 61. Eléments des menus contextuels de Property

Nom	Description
Encapsuler dans OR	Encapsule l'élément Property dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Property dans l'élément And Rule Group.
Nouvel argument	Permet d'ajouter un nouvel argument à l'élément de règle de propriété.
Supprimer l'argument	Permet de supprimer un argument de l'élément de règle de propriété.

ATTENTION :

Depuis Cúram V6, CER et le gestionnaire de dépendances prennent en charge le recalcul automatique des valeurs calculées par CER si leurs dépendances changent.

Si vous modifiez l'implémentation d'une méthode de propriété, CER et le gestionnaire de dépendances ne sauront *pas* recalculer automatiquement les valeurs d'attributs calculées à l'aide de l'ancienne version de votre méthode de propriété.

Une fois qu'une méthode de propriété a été utilisée dans un environnement de production pour les valeurs d'attributs stockées, au lieu de modifier l'implémentation, créez une nouvelle méthode de propriété (avec la nouvelle implémentation requise), et modifiez vos jeux de règles pour utiliser la nouvelle méthode de propriété. Lorsque vous publiez les modifications apportées à votre jeu de règles pour pointer vers la nouvelle méthode de propriété, CER et le gestionnaire de dépendances recalculeront automatiquement toutes les instances des valeurs d'attributs affectées.

7.4.5 Custom Expression

L'élément Custom Expression fournit une représentation graphique d'un noeud XML valide défini par l'utilisateur. L'élément Custom Expression peut être ajouté à un élément dans CER, et l'utilisateur a l'obligation de s'assurer que les noms de Custom Expression ne correspondent pas à des noms de noeuds du langage CER.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 62. Eléments des menus contextuels de l'élément Custom Expression

Nom	Description
Editer une expression personnalisée	Affiche la fenêtre contextuelle permettant d'éditer l'expression personnalisée.

7.4.6 Existence Timeline

L'élément Existence Timeline fournit une représentation graphique de l'expression 'existencetimeline'. D'autres éléments de la palette (Date ou Rule Reference) peuvent être ajoutés aux valeurs FromDate et ToDate de l'élément Existence Timeline. D'autres éléments de la palette (Date ou Rule Reference) peuvent

être ajoutés aux valeurs preExistenceValue, postExistenceValue et ExistenceValue de l'élément Existence Timeline. L'élément Existence Timeline peut être ajouté à d'autres éléments de la palette, par exemple, Repeating Rule.

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 63. Eléments de propriétés de Existence Timeline

Nom	Description
Type de données	Le type de données de la chronologies d'existence. Si l'utilisateur souhaite remplacer le type de données par une chronologie, cochez la case correspondante. Si l'utilisateur souhaite remplacer le type de données par une liste, cochez la case correspondante. Cet élément est visible dans l'onglet Métier.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 64. Eléments des menus contextuels de l'élément Existence Timeline

Nom	Description
Editer une chronologies d'existence	Fournit 10 types de données (Chaîne, Nombre, Booléen, Date, Date-heure, Entrée de table de codes, Classe de règles, Classe Java, Liste et Message) pour un type d'intervalle de l'élément de règle de la chronologies d'existence.

7.4.7 Timeline

L'élément Timeline fournit une représentation graphique de l'expression 'Timeline'. Une valeur initiale peut être définie pour un élément Timeline à l'aide de l'option de menu qu'il comporte. D'autres éléments de la palette, par exemple, Interval, FixedList, Repeating Rule, etc., peuvent être ajoutés à l'élément Timeline.

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 65. Eléments de propriétés d'Existence Timeline

Nom	Description
Type de données	Le type de données de l'élément Timeline. Si l'utilisateur souhaite remplacer le type de données par une chronologie, cochez la case correspondante. Si l'utilisateur souhaite remplacer le type de données par une liste, cochez la case correspondante. Cet élément est visible dans l'onglet Métier.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 66. Eléments des menus contextuels de l'élément Timeline

Nom	Description
Ajouter intervalles	Ajoute des intervalles à la chronologie
Ajouter une valeur initiale	Ajoute une valeur initiale à la chronologie
Retirer des intervalles	Supprime des intervalles de la chronologie
Retirer une valeur initiale	Supprimer la valeur initiale de la chronologie
Editer la chronologie	Affiche l'assistant de chronologie pour éditer cette dernière

7.4.8 Interval

L'élément Interval fournit une représentation graphique de l'expression 'Interval'. D'autres éléments de la palette (Date or Reference) peuvent être ajoutés aux éléments FromDate et ToDate de l'élément Existence Timeline. L'élément Interval peut uniquement être ajouté à l'élément Intervals d'un élément Timeline. Le type d'intervalle peut être défini à l'aide de l'assistant d'intervalle. L'élément Interval contient un élément de début et un élément de valeur comme éléments enfant.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 67. Eléments des menus contextuels de l'élément Interval

Nom	Description
Editer l'intervalle	Affiche l'assistant permettant d'éditer l'intervalle

7.4.9 CombineSuccessionSet

L'élément CombineSuccessionSet fournit une représentation graphique de l'expression 'combineSuccessionSet'. D'autres éléments de la palette (Filter ou Fixed List) peuvent être ajoutés à l'élément CombineSuccessionSet. L'élément CombineSuccessionSet peut être ajouté à d'autres éléments de la palette, par exemple, When ou Create.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 68. Eléments des menus contextuels de l'élément CombineSuccessionSet

Nom	Description
Editer CombineSuccessionSet	Permet d'éditer l'élément CombineSuccessionSet en sélectionnant la classe et le jeu de règles auxquels vous souhaitez faire référence.

7.4.10 Call

L'élément Call fournit une représentation graphique de l'expression 'call' et appelle une méthode Java statique pour exécuter un calcul complexe. Le nouvel argument peut être ajouté à l'élément Call. D'autres éléments de la palette, par exemple Date, Period Length ou Rule Reference peuvent être ajoutés à l'argument de l'élément Call. L'élément Period Length peut être ajouté à d'autres éléments de la palette, par exemple, Create.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 69. Eléments de propriétés de Call

Nom	Description
Classe	Le nom de la classe disposant de la méthode appelante. Cet élément est visible dans l'onglet technique.
Nom de méthode	Le nom de la méthode qui sera appelée. Cet élément est visible dans l'onglet technique.

Tableau 69. Eléments de propriétés de Call (suite)

Nom	Description
Type de date	Le type de renvoi de l'appel. Il doit être identique au type de données de l'attribut. Si l'utilisateur souhaite remplacer le type de données par une chronologie, cochez la case correspondante. Si l'utilisateur souhaite remplacer le type de données par une liste, cochez la case correspondante. Cet élément est visible dans l'onglet Métier.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 70. Eléments des menus contextuels de Call

Nom	Description
Encapsuler dans AND	Encapsule l'élément de règle d'appel dans l'élément And Rule Group.
Encapsuler dans OR	Encapsule l'élément de règle d'appel dans l'élément Or Rule Group.
Nouvel argument	Permet d'ajouter un nouvel argument à l'élément de règle d'appel.
Supprimer l'argument	Permet de supprimer un argument de l'élément de règle d'appel.

ATTENTION :

Depuis Cúram V6, CER et le gestionnaire de dépendances prennent en charge le recalcul automatique des valeurs calculées par CER si leurs dépendances changent.

Si vous modifiez l'implémentation d'une méthode statique, CER et le gestionnaire de dépendances ne sauront *pas* recalculer automatiquement les valeurs d'attributs calculées à l'aide de l'ancienne version de votre méthode statique.

Une fois qu'une méthode statique a été utilisée dans un environnement de production pour les valeurs d'attributs stockées, au lieu de modifier l'implémentation, créez une nouvelle méthode statique (avec la nouvelle implémentation requise), et modifiez vos jeux de règles pour utiliser la nouvelle méthode statique. Lorsque vous publiez les modifications apportées à votre jeu de règles pour pointer vers la nouvelle méthode statique, CER et le gestionnaire de dépendances recalculeront automatiquement toutes les instances des valeurs d'attributs affectées.

7.4.11 Period Length

L'élément Period Length fournit une représentation graphique de l'expression 'periodlength' et calcule la quantité d'unités de temps entre deux dates. D'autres de la palette, comme Date, Call ou Rule Reference, peuvent être ajoutés à l'élément Period Length. L'élément Period Length peut être ajouté à d'autres éléments de la palette, par exemple, Create.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 71. Eléments de propriétés de Period Length

Nom	Description
Unité	Fournit quatre types d'unités (jours, semaines, mois et années) pour l'élément de règle Period Length. Cet élément est visible dans l'onglet technique.
EndDateInclusion	Fournit deux types d'inclusions de date (inclusif ou exclusif). Cet élément est visible dans l'onglet technique.

7.4.12 ALL

L'élément ALL fournit une représentation graphique de l'expression 'all' et renvoie une valeur booléenne. Il opère sur une liste de valeurs booléennes pour déterminer si toutes les valeurs de la liste sont vraies. L'élément ALL ne comporte pas d'élément fixedlist. D'autres éléments de la palette (Repeating Rule ou Fixed List) peuvent être ajoutés à l'élément ALL. L'élément ALL peut être ajoutés à d'autres éléments de la palette, par exemple, Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 72. *Éléments des menus contextuels de l'élément ALL*

Nom	Description
Encapsuler dans OR	Encapsule l'élément And Rule Group dans un autre élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément And Rule Group dans un autre élément And Rule Group.
Remplacer par OR	Remplace l'élément And Rule Group par l'élément Or Rule Group.

7.4.13 ANY

L'élément ANY fournit une représentation graphique de l'expression 'any' et renvoie une valeur booléenne. Il opère sur une liste de valeurs booléennes pour déterminer si des valeurs de la liste sont vraies. L'élément ANY ne comporte pas de fixedlist. D'autres éléments de la palette (Repeating Rule ou Fixed List) peuvent être ajoutés à l'élément ANY. L'élément ANY peut être ajouté à d'autres éléments de la palette, par exemple, Repeating Rule.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 73. *Éléments des menus contextuels de l'élément ANY*

Nom	Description
Encapsuler dans OR	Encapsule l'élément Or Rule Group dans un autre élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Or Rule Group dans l'élément And Rule Group.
Remplacer par AND	Remplace l'élément Or Rule Group par l'élément And Rule Group.

7.4.14 This

L'élément This fournit une représentation graphique de l'expression 'this' correspondant à une référence à l'objet de règle en cours. L'élément This peut être ajouté à d'autres éléments de la palette, par exemple, When ou Any. Aucun élément ne peut être ajouté à l'élément This.

7.4.15 Sort

L'élément Sort fournit une représentation graphique de l'expression 'sort'. Celui-ci utilise une liste comme élément enfant et effectue un tri sur celle-ci.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 74. *Éléments des menus contextuels de l'élément ANY*

Nom	Description
Nouveaux éléments Tri croissant	Permet d'ajouter un élément de tri croissant.

Tableau 74. Eléments des menus contextuels de l'élément ANY (suite)

Nom	Description
Nouveaux éléments Tri décroissant	Permet d'ajouter un élément de tri décroissant.

7.4.16 Shared Rule Reference

L'élément Shared Rule Reference correspond essentiellement à une référence normale comportant un élément Create. Il présente un assistant qui affiche les noms des classes de règles comportant un attribut principal défini dans le jeu de règles en cours. L'élément Shared Rule Reference peut être modifié en sélectionnant l'option de menu "Editer une référence de règle partagée" du diagramme. Une règle partagée représente une classe comportant un attribut principal. L'assistant de règles partagées permet à l'utilisateur de créer des règles partagées pouvant être utilisées pour la création des références de règles partagées.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 75. Eléments de propriétés de Shared Rule Reference

Nom	Description
Classe	Le nom de la classe de règles. Cet élément est visible dans l'onglet Métier.
Attribut	Le nom de l'attribut. Cet élément est visible dans l'onglet Métier.
Élément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Cet élément est activé lorsque l'option Élément unique est sélectionnée.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return last) lorsque plusieurs éléments sont trouvés. Cet élément est activé lorsque l'option Élément unique est sélectionnée.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 76. Eléments des menus contextuels de l'élément Shared Rule Reference

Nom	Description
Encapsuler dans OR	Encapsule l'élément Shared Rule Reference dans l'élément Or Rule Group.
Encapsuler dans AND	Encapsule l'élément Shared Rule Reference dans l'élément And Rule Group.
Editer la référence	Permet d'éditer l'élément de référence en sélectionnant la règle à laquelle vous souhaitez faire référence.
Editer une référence de règle partagée	Permet d'éditer l'élément de référence de règle partagée en sélectionnant la règle partagée à laquelle vous souhaitez faire référence. Consultez la boîte de dialogue d'édition de la règle partagée ci-après.
Nouveau paramètre	Permet de créer un nouveau paramètre en sélectionnant l'attribut auquel vous souhaitez ajouter un paramètre. Consultez la boîte de dialogue Nouveau paramètre ci-après.
Nouveau paramètre obligatoire	Permet de créer un paramètre obligatoire.

7.4.17 CONCAT

L'élément Concat fournit une représentation graphique de l'expression 'concat' et crée un message localisable en concaténant une liste de valeurs. L'élément Concat comporte un élément fixedlist contenant des objets de chaîne. D'autres éléments de la palette (String ou Rule Reference) peuvent être ajoutés au membre vide (fixedlist) de l'élément Concat. L'élément Concat peut être ajouté à d'autres éléments de la palette, par exemple, When ou Repeating Rule.

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 77. Eléments de propriétés Concat

Nom	Description
Type de données	Le type de données de l'élément Concat. Celui-ci doit correspondre au type de données de l'attribut. Si l'utilisateur souhaite remplacer le type de données par une chronologie, cochez la case correspondante. Si l'utilisateur souhaite remplacer le type de données par une liste, cochez la case correspondante. Cet élément est visible dans l'onglet Métier.

7.4.18 Join Lists

L'élément JoinLists fournit une représentation graphique de l'expression 'joinlists' et crée une nouvelle liste en joignant plusieurs listes existantes. D'autres éléments de la palette (Rule Reference ou Choose) peuvent être ajoutés au membre vide (fixedlist) de l'élément JoinLists. L'élément JoinLists peut être ajouté à d'autres éléments de la palette, par exemple, And Rule Group, Or Rule Group ou When.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 78. Eléments de propriétés Join Lists

Nom	Description
Élément unique	Seul élément renvoyé depuis l'élément.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return first, return last) lorsque plusieurs éléments sont trouvés.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 79. Eléments des menus contextuels Join Lists

Nom	Description
Retirer des doublons	Permet de supprimer les éléments en double dans l'élément Join Lists.
Concaténer des résultats	Permet de concaténer les éléments dans l'élément Join Lists.
Joindre des listes internes	Permet d'associer les listes pour n'en former qu'une.

7.5 Référence des éléments de règle pour les modèles d'unités du foyer

7.5.1 Composition du foyer

Ce modèle de composition est utilisé dans les règles Curam Global Income Support (CGIS) et présente une composition de foyer.

7.5.2 Catégorie de foyer

Ce modèle représente une nouvelle catégorie de foyer dans les règles CGIS.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 80. Eléments des menus contextuels de la catégorie de foyer

Nom	Description
Ajouter un membre obligatoire	Permet d'ajouter un membre obligatoire à l'élément de règle de la catégorie de foyer.
Retirer un membre obligatoire	Permet de supprimer un membre obligatoire de l'élément de règle de la catégorie de foyer.
Ajouter un membre facultatif	Permet d'ajouter un membre facultatif à l'élément de règle de la catégorie de foyer.
Retirer un membre facultatif	Permet de supprimer un membre facultatif de l'élément de règle de la catégorie de foyer.

7.6 Référence des éléments de règle pour les modèles d'unités financières

7.6.1 Unité financière

Ce modèle est utilisé dans les règles Curam Global Income Support et présente une unité financière.

Figure 30. Menu contextuel :

7.6.2 Catégorie d'unité financière

Ce modèle représente une nouvelle catégorie d'unité financière pour une unité financière dans les règles CGIS

7.6.3 Membre d'unité financière

Ce modèle représente un nouveau membre d'unité financière pour une unité financière dans les règles CGIS

7.7 Référence des éléments de règle pour les modèles d'unités d'assistance alimentaire

7.7.1 Unité d'assistance alimentaire

Ce modèle représente une nouvelle unité d'assistance alimentaire dans les règles CGIS.

Figure 31. Menu contextuel :

Un diagramme d'assistance alimentaire doit tout d'abord être configuré avant d'être entièrement édité. Certains assistants de l'éditeur nécessitent que la configuration soit définie avant de fournir des options de menu spécifiques à l'assistance alimentaire, par exemple, l'assistant de référence. La configuration d'un diagramme d'assistance alimentaire peut être modifiée à tout moment.

7.7.2 Catégorie d'assistance alimentaire pour une personne seule

Ce modèle représente une nouvelle catégorie d'assistance alimentaire pour une personne seule dans les règles CGIS.

7.7.3 Catégorie d'assistance alimentaire pour plusieurs personnes

Ce modèle représente une nouvelle catégorie d'assistance alimentaire pour plusieurs personnes dans les règles CGIS.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 81. Eléments des menus contextuels de la catégorie d'assistance alimentaire pour plusieurs personnes

Nom	Description
Supprimer le groupe de repas	Supprime un groupe de repas de l'élément de règle de la catégorie d'assistance alimentaire pour plusieurs personnes.
Retirer des proches	Supprimer des proches de l'élément de règle de la catégorie d'assistance alimentaire pour plusieurs personnes.
Retirer des annulations	Supprime une annulation de l'élément de règle de la catégorie d'assistance alimentaire pour plusieurs personnes.
Retirer le chef des membres du foyer	Supprimer le chef de famille de l'élément de règle de la catégorie d'assistance alimentaire pour plusieurs personnes.
Retirer un membre facultatif	Supprimer un membre facultatif de l'élément de règle de la catégorie d'assistance alimentaire pour plusieurs personnes.

7.7.4 Membres de groupe de repas

Ce modèle représente les nouveaux membres de groupe de repas de l'assistance alimentaire dans les règles CGIS.

7.7.5 Proches

Ce modèle représente de nouveaux proches de l'assistance alimentaire dans les règles CGIS.

7.7.6 Annuler

Ce modèle représente une nouvelle annulation d'assistance alimentaire dans les règles CGIS.

7.7.7 Membre du foyer

Ce modèle représente le nouveau membre du foyer de l'assistance alimentaire dans les règles CGIS.

7.7.8 Membres facultatifs

Ce modèle représente les nouveaux membres facultatifs d'assistance alimentaire dans les règles CGIS.

7.7.9 Exceptions

Ce modèle représente de nouvelles exceptions d'assistance alimentaire dans les règles CGIS.

7.8 Référence des éléments de règle pour les modèles de table de décision

7.8.1 Decision Table

L'élément Decision Table fournit une représentation graphique de l'expression 'decision table'. Lorsqu'un élément Decision Table a été déplacé vers une règle ou un attribut, l'assistant **Créer une table de décision** s'affiche. L'utilisateur doit définir les options suivantes :

- Nombre de lignes - il s'agit du nombre de lignes contenues dans la table de décision, le maximum étant de 99 ;
- Type de résultat - il s'agit du type de retour de la table de décision, le type de résultat doit correspondre au type de résultat de la règle ou l'attribut contenant la table de décision ;
- Classe de règles - les utilisateurs peuvent sélectionner la classe de règles en cours ou modifier les classes de règles.

Lorsque l'utilisateur clique sur le bouton *Suivant*, il peut utiliser une règle existante, ou choisir de créer une nouvelle règle.

Propriétés :

Le tableau suivant répertorie les éléments de propriétés spécifiques de cet élément :

Tableau 82. Eléments de propriétés de table de décision

Nom	Description
Classe	Le nom de la classe de règles choisie comme type. Cet élément est visible dans l'onglet Métier.
Attribut	Le nom de l'attribut contenant la table de décision. Cet élément est visible dans l'onglet Métier
Élément unique	Seul élément renvoyé depuis l'élément. Cet élément est visible dans l'onglet technique.
Comportement lorsqu'aucun élément n'est trouvé.	Renvoie l'un de ces résultats (error, return null) lorsqu'aucun élément n'est trouvé. Celui-ci est activé lorsque l'option 'Élément unique' est sélectionnée.
Comportement lorsque plusieurs éléments sont trouvés.	Renvoie l'un de ces résultats (error, return null, return first, return last) lorsque plusieurs éléments sont trouvés. Celui-ci est activé lorsque l'option 'Élément unique' est sélectionnée.

Menu contextuel :

Le tableau suivant répertorie les éléments de menu contextuel de cet élément :

Tableau 83. Eléments des menus contextuels de l'élément Decision Table

Nom	Description
Editer la table de décision	Modifie le type de résultat ou l'attribut associé.
Ajouter une nouvelle ligne.	Ajoute une nouvelle ligne à la table de décision.

Chapitre 8. CER Best Practices

8.1 Introduction

Ce chapitre décrit les meilleures pratiques à suivre lors de l'écriture des jeux de règles CER. L'application de ces meilleures pratiques facilitera le développement, le test et la gestion de vos jeux de règles.

8.2 Attribut de règle description

Chaque classe de règles hérite enfin d'une «classe de règles racine» CER. Cette classe racine inclut un attribut de règle description qui dispose d'une implémentation de valeur par défaut (mais pas particulièrement utile).

La valeur description d'un objet de règle est sortie dans RuleDoc, ainsi que par la méthode toString sur RuleObject (que de nombreux IDE utilisent lorsque vous «cliquez» sur une variable). Par conséquent, attribut de règle description significatif peut être indispensable lors de l'explication du comportement de votre jeu de règles.

Vous devez remplacer le calcul description par défaut en créant de façon explicite un attribut description sur chacune de vos classes de règles. Vous pouvez utiliser l'éditeur CER pour créer un attribut description comme la création d'un attribut normal sur une classe de règles. Le valeur du jeu de règles CER signale un avertissement si vous disposez d'une classe de règles qui ne définit pas (ou hérite d'une autre classe de règles définie) un attribut de règle description.

L'attribut description est un message localisable et (tout comme d'autres attributs de règle) son calcul peut être simple ou aussi complexe que nécessaire.

Voici un exemple de jeu de règles, dans lequel certaines classes de règles fournissent une implémentation de description :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_description"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="lastName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Remplacement de la description par défaut -->
    <Attribute name="description">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Concaténation du prénom et du nom de la personne -->
        <concat>
          <fixedlist>
            <listof>
              <javaclass name="Object"/>
            </listof>
            <members>

              <reference attribute="firstName"/>
              <String value=" "/>
              <reference attribute="lastName"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Income">
    <!-- Personne à laquelle cet
    enregistrement de revenu est associé. -->
    <Attribute name="person">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
    <Attribute name="startDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>

```

Voici une classe de test qui crée des objets de règle (Person, Income et Benefit) :

```

package curam.creole.example;

import java.io.File;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.SessionDoc;
import curam.creole.execution.session.Session_Factory;
import
    curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.ruleclass.Example_description.impl.Benefit;
import
    curam.creole.ruleclass.Example_description.impl.Benefit_Factory;
import curam.creole.ruleclass.Example_description.impl.Income;
import
    curam.creole.ruleclass.Example_description.impl.Income_Factory;
import curam.creole.ruleclass.Example_description.impl.Person;
import
    curam.creole.ruleclass.Example_description.impl.Person_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.util.type.Date;

/**
 * Test de la description de l'attribut de règle.
 */
public class TestDescription extends TestCase {

    /**
     * Test de la description de l'attribut de règle.
     */
    public void testDescriptions() {

        /**
         * Création d'une nouvelle session.
         */
        final Session session =
            Session_Factory.getFactory().newInstance(
                new RecalculationsProhibited(),
                new InMemoryDataStorage(
                    new StronglyTypedRuleObjectFactory()));

        /**
         * Création SessionDoc pour produire un rapport sur les objets de règle.
         */
        final SessionDoc sessionDoc = new SessionDoc(session);

        /**
         * Création d'un objet de règle Person.
         */
        final Person person =
            Person_Factory.getFactory().newInstance(session);
        person.firstName().specifyValue("John");
        person.lastName().specifyValue("Smith");

        /**
         * Création d'un objet de règle Income.
         */
        final Income income =
            Income_Factory.getFactory().newInstance(session);
        income.person().specifyValue(person);
        income.amount().specifyValue(123);
        income.startDate().specifyValue(Date.fromISO8601("20070101"));

        /**
         * Création d'un objet de règle Benefit.
         */
        final Benefit benefit =
            Benefit_Factory.getFactory().newInstance(session);
        benefit.person().specifyValue(person);
        benefit.amount().specifyValue(234);

        /**

```

Lorsque le test est exécuté, il génère cette sortie, qui indique les descriptions d'objet de règle :

```
John Smith  
John Smith's income, starting on 01/01/07 00:00  
Undescribed instance of rule class 'Benefit', id '3'
```

A la fin du test, les objets de règle de la session sont sortis en tant que SessionDoc. Le récapitulatif SessionDoc de niveau supérieur affiche les objets de règle créés et répertorie la description de chaque objet de règle :

Example_description

Generated: 13-Jul-2012 11:52:43

External rule objects

Details	Type	Description	Action
details	Example_description.Benefit	Undescribed instance of rule class 'Benefit', id '3'	Created during this session
details	Example_description.Income	John Smith's income, starting on 01/01/07 00:00	Created during this session
details	Example_description.Person	John Smith	Created during this session

Internal rule objects

Details	Type	Description	Action
---------	------	-------------	--------

Figure 32. SessionDoc indiquant les valeurs description de l'objet de règle

La description de l'objet de règle Benefit est la description par défaut. En cas d'absence d'une bonne implémentation de description, une personne lisant SessionDoc devra peut-être parcourir SessionDoc pour donner un sens à l'objet de règle Benefit :

Rule Object

Generated: 13-Jul-2012 11:52:43

Type

Example_description.Benefit

Description

Undescribed instance of rule class 'Benefit', id '3'

Creation

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by
amount	Number	SPECIFIED	234	<ul style="list-style-type: none">Specified externally.	None	None
description	Message	CALCULATED	Undescribed instance of rule class 'Benefit', id '3'	<ul style="list-style-type: none">Default rule object description.	None	None
person	Person	SPECIFIED	John Smith	<ul style="list-style-type: none">Specified externally.	None	None

Figure 33. SessionDoc pour un objet de règle sans remplacement d'élément description

Cette capture d'écran présente comment un environnement de développement intégré (tel qu'Eclipse) utilise la méthode toString d'un objet lors du débogage, qui calcule (pour les objets de règle) l'élément description :

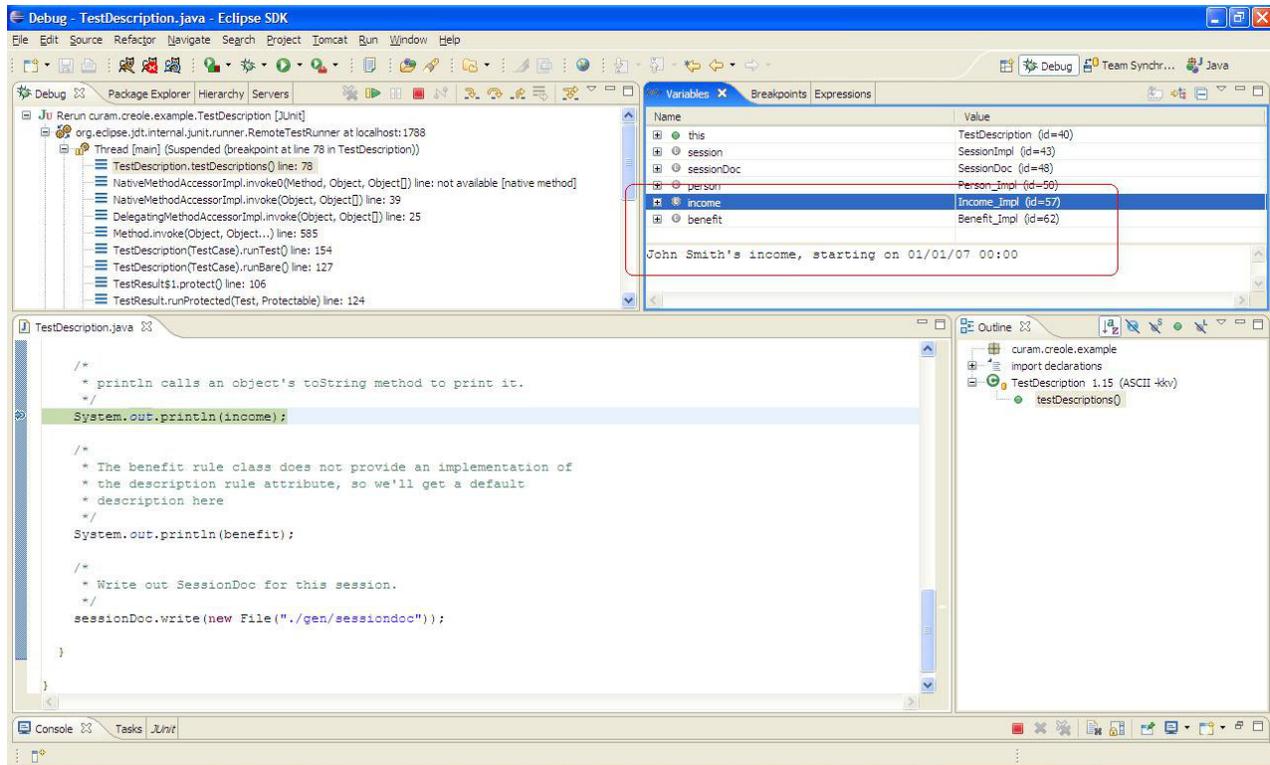


Figure 34. Utilisation d'un élément description dans un environnement de développement intégré

Conseil : N'oubliez pas que le but de l'élément description est de décrire une instance d'objet de règle, et non la classe de règles proprement dite.

En particulier, le calcul de votre attribut de règle description doit inclure des données indiquant qu'il est facile de faire la distinction entre différentes instances d'objet de règle de votre classe de règles.

8.3 Obtention d'un jeu de règles rapidement fonctionnel

Il peut être utile d'obtenir d'un jeu de règles CER rapidement fonctionnel en retardant certaines tâches de développement de règles.

Pensez à utiliser un ou plusieurs des raccourcis suivants :

- Créez des classes de règles vides (par ex., pas d'attributs de règle) pour chaque concept métier. Les attributs de règle pourront être ajoutés ultérieurement. Par exemple, vous pouvez utiliser l'éditeur CER pour créer une classe de règles vide et ajoutez-y un attribut ultérieurement.
- Créez des dérivations codées en dur pour les attributs de règles ; les règles métier peuvent être ajoutées ultérieurement. Par exemple, la déclaration du calcul d'un attribut de règle `isEligible` sur `<true>` peut vous permettre d'écrire des tests et/ou d'intégrer CER à votre propre application. Vous pouvez remplacer la dérivation "toujours éligible" codée en dur par les véritables règles métier ultérieurement. Par exemple, vous pouvez utiliser l'éditeur CER pour définir l'élément de règle "Boolean" (sa valeur par défaut est définie sur `true`) en attribut de règle `isEligible`.
- Créez des messages en tant que chaînes codées en dur, à environnement local unique et en texte clair. Vous pouvez convertir ultérieurement les chaînes en messages localisables. Par exemple, vous pouvez éditer l'éditeur CER pour définir l'élément de règle "Resource Message" ou "XML Message" en attribut de règle.

- Utilisez les valeurs de chaîne au lieu des valeurs de table de codes d'application. Vous pouvez convertir les chaînes en codes ultérieurement (et mettre à jour les règles qui effectuent les tests pour elles). Par exemple, vous pouvez utiliser l'éditeur CER pour définir l'élément de règle "String" en attribut de règle.

Important : La prise en compte de ces raccourcis ne vous abstient *pas* d'effectuer le travail de façon plus rigoureuse. Une partie du travail est simplement retardée jusqu'au stade auquel votre jeu de règles est "exécutable".

Vous ne devez *pas* interrompre la création de tests pour vos règles ; en particulier, si vous prenez en compte ces raccourcis, un jeu de tests approprié vous permettra d'introduire certains types d'erreur lorsque vous annulez ces raccourcis. Créez vos tests au fur et à mesure de l'écriture de vos règles.

Vous pouvez également être tenté de retarder la création d'attributs de règle description pour vos classes de règles ; toutefois, dans les premières étapes de la conception du jeu de règles, une telle stratégie peut s'avérer risquée, car les attributs de règle description peuvent vous être d'une grande aide dans le cadre du débogage des règles, pour un coût relativement bon marché.

8.4 Nommage des éléments de règle

Les attributs de règle CER doivent être nommés pour décrire leur signification métier. Nommez un attribut de règle en fonction du résultat qu'il permet d'obtenir, et non en fonction de la manière dont l'attribut de règle le fournit.

Un nom de jeu de règles peut être une combinaison de lettres (non accentuées, dans la plage de caractères A-Z standard), de nombres et de traits de soulignement. Un nom de jeu de règles doit commencer par une lettre majuscule.

Un nom de classe peut être une combinaison de lettres (non accentuées, dans la plage de caractères A-Z standard), de nombres et de traits de soulignement. Un nom de classe doit commencer par une lettre majuscule.

Un nom d'attribut peut être une combinaison de lettres (non accentuées, dans la plage de caractères A-Z standard), de nombres et de traits de soulignement. Un nom d'attribut doit commencer par une lettre minuscule.

Conseil : Utilisez "camelCase" pour nommer vos éléments de règle.

La casse mixte (camelCase) est la pratique d'écriture des mots ou des phrases composés dans lesquels les éléments sont joints sans espaces, avec la lettre initiale de chaque élément dans l'enceinte et la première lettre en majuscules ou minuscules.

Remarque : Chaque élément de règle dispose d'un nom unique qui ne peut pas être localisé. Pour obtenir des descriptions localisables, utilisez l'annotation "Label", comme décrit dans 3.3.2, «Localisation des descriptions d'artefacts de règles CER», à la page 13.

8.5 Moments opportuns à l'utilisation de l'expression reference

L'utilisation correcte de l'expression reference est essentielle à la structure d'un bon jeu de règles CER. L'utilisation de l'expression reference va de pair avec la création du bon nombre d'attributs de règle. L'éditeur CER fournit différents types de scénario pour créer et utiliser l'élément de référence de règle. Voir l'élément "rule" dans 7.2.1, «Rule», à la page 111.

Trouver le bon équilibre (entre une utilisation insuffisante et une utilisation abusive d'expressions reference) ne relève pas d'une science exacte. Toutefois, voici quelques instructions générales :

- Si vous constatez que certaines de vos expressions sont imbriquées très profondément ou sont très complexes, un nombre insuffisant d'expressions est défini. Pensez à scinder les expressions complexes en créant des attributs de règle pour un bloc d'expressions significatif et en utilisant à la place une expression reference dans le nouvel attribut de règle.
- Si vos exigences ont un concept fort ou un calcul qui ne correspond pas parfaitement à un attribut de règle, pensez à créer un attribut de règle de ce type.
- Si plusieurs expressions répètent le même type de calcul, votre jeu de règles peut bénéficier de la création d'un attribut de règle pour implémenter la logique commune.
- Si vous trouvez qu'un attribut de règle est difficile à nommer, l'attribut de règle peut être une encapsulation de logique obsolète et votre jeu de règles comporte un nombre trop important d'expressions reference. Pensez à supprimer l'attribut de règle et à "mettre en ligne" sa dérivation dans les endroits où il est utilisé, surtout si l'attribut de règle n'est référencé qu'à partir d'un autre calcul.

8.6 Utilisation de l'outil RuleDoc

Lorsque vos jeux de règles CER sont petits en taille, vous pouvez les ouvrir directement dans l'éditeur CER et les comprendre facilement dans leur intégralité.

Toutefois, au fur et à mesure que votre jeu de règles augmente en complexité, vous pouvez obtenir des informations sur la structure et le comportement de votre jeu de règles via l'outil RuleDoc de CER.

Voir 3.10, «RuleDoc», à la page 20 pour savoir comment générer un outil RuleDoc à partir de vos jeux de règles CER.

8.7 Normalisation des règles communes

Lorsque vous développez votre jeu de règles, vous pouvez remarquer des règles qui sont similaires sur différentes parties de votre fonctionnalité de jeu de règles.

Veillez à identifier les règles communes et à les centraliser.

En règle générale, deux options sont disponibles lors de la centralisation des règles communes :

- **Héritage**

Utilisez la prise en charge CER dans le cadre de l'héritage de l'implémentation pour permettre à une classe de règles d'en étendre une autre. L'éditeur CER fournit le mécanisme d'héritage permettant de concevoir la règle. Voir l'élément "extends" dans 6.6.3, «Propriétés de classe de règles», à la page 108.

- **Confinement**

Utilisez la prise en charge CER pour créer de nouveaux objets de règle permettant à une classe de règles de créer de nouvelles instances d'une autre classe de règles lorsque cela est nécessaire. L'éditeur CER fournit le mécanisme de confinement permettant de concevoir la règle. Voir la section change rule set and rule class dans 6.7, «Assistants d'éléments de règles», à la page 109.

Parfois, il peut être difficile d'identifier le mécanisme à utiliser lors de la centralisation des règles communes. En règle générale, vous devez utiliser l'héritage avec prudence et uniquement lorsque la classe de sous-règle représente un concept métier qui "est" véritablement une instance du concept métier représentée par la super-classe. En particulier, CER ne prend pas en charge l'héritage multiple.

Comme exemple d'héritage, citons une personne possédant des ressources, chacune d'elles étant un bâtiment ou un véhicule. Les classes de règles Building et Vehicle étendent chacune une classe de règles Resource abstraite. Voir la liste dans 4.4.1.1, «Classes de règles», à la page 40.

Vous devez utiliser le confinement lorsque le concept métier représenté par une classe de règles "a une" instance du concept métier représentée par la classe de règles confinée.

Comme exemple de confinement, citons une personne pour laquelle plusieurs tests de plage d'âge sont appliqués. La classe de règles Person crée des instances de AgeRangeTest.

Si vous constatez que des classes de règles similaires se trouvent dans différents jeux de règles, pensez à utiliser les fonctions CER (depuis Cúram V6) pour permettre à un jeu de règles de référencer des artefacts dans un autre. Placez les classes de règles communes dans un ou plusieurs jeux de règles et placez les classes de règles qui ne sont pas communes dans d'autres jeux de règles.

8.8 Suppression des règles inutilisées

Lorsque vous centralisez des règles communes, vous trouverez peut-être que certains attributs de règles ne sont plus référencés dans d'autres calculs et peuvent être supprimés de votre jeu de règles (dans le cadre de l'exercice de "débroussaillage").

CER inclut la prise en charge pour la production de rapports d'attributs de règles qui ne sont pas référencés à partir d'autres calculs de votre jeu de règles et sont, par conséquent, éligibles à la suppression. Vous pouvez également utiliser l'éditeur CER pour supprimer les classes de règles et les attributs de règles. Voir 6.2.4, «Vue technique», à la page 101.

Pour exécuter le rapport d'attribut CER non utilisé, voir 3.12, «Attributs de règles inutilisés», à la page 27.

avertissement : Notez qu'il est parfaitement possible à un attribut de règle d'être un attribut de règle de "niveau supérieur" qui est uniquement référencé à partir d'un code client. Ces attributs peuvent être signalés comme "non utilisés" par ce rapport, mais tout attribut de règle apparemment inutilisé ne doit pas être supprimé de votre jeu de règles à moins de vérifier qu'aucun code client ou test n'en dépende.

8.9 Ordre des déclarations

En règle générale, l'ordre des déclarations dans vos jeux de règles n'a aucun effet sur le comportement.

8.9.1 Ordre des classes de règles dans un jeu de règles

Vous pouvez réorganiser les classes de règles de votre jeu de règles dans l'ordre qui vous est le plus utile. La réorganisation de classes de règles n'a aucun effet sur le comportement de vos jeux de règles.

8.9.2 Ordre des attributs de règles calculés dans une classe de règles

De même, vous pouvez réorganiser les attributs de règles *calculés* de votre classe de règles dans l'ordre qui vous est le plus utile. La réorganisation d'attributs de règles *calculés* n'a aucun effet sur le comportement de vos jeux de règles.

8.9.3 Ordre des attributs initialisés dans une classe de règles

Chaque fois qu'une instance d'objet de règle de la classe est créée (dans les règles à l'aide de l'expression `create` ou via le code Java à l'aide des classes de règles générées ou l'API de règle dynamique), les valeurs de tous les attributs initialisés doivent être spécifiés *dans leur ordre de définition dans la classe de règles*.

avertissement : Dans ce cas, évitez de réorganiser les attributs dans un bloc `Initialization`, à moins que vous ne soyez préparé à mettre à jour également tous les espaces (dans les règles ou le code Java) qui créent des instances d'objet de règle de la classe de règles.

8.9.4 Ordre des conditions booléennes

L'ordre des conditions booléennes dans une expression `all` ou `any` n'affecte pas la valeur logique du résultat.

Toutefois, lors de l'exécution, le traitement des conditions booléennes s'arrête dès qu'un résultat est confirmé ; pensez à l'ordre des conditions booléennes afin que l'expression `all` ou `any` obtienne son résultat le plus rapidement possible.

Explication :

- pour les expressions `all`, en plaçant des conditions booléennes susceptibles d'être évaluées *false* plus tôt dans la liste ; et
- pour les expressions `any`, en plaçant des conditions booléennes susceptibles d'être évaluées *true* plus tôt dans la liste.

Vous pouvez utiliser l'éditeur CER pour modifier l'ordre des éléments de règles booléennes dans l'élément de règle `Any`. Par exemple, organisez tous les éléments de règles booléennes avec, d'abord, la valeur `true`.

8.10 Création d'objets de règle

Pensez attentivement à la création de vos objets de règle.

En particulier, si vous utilisez CER dans votre propre application, définissez rapidement les objets de règle créés par votre code d'application et les objets de règle créés par vos règles.

Voir 4.3, «Objets de règle externes et internes», à la page 33 pour plus d'informations.

8.11 Transmission d'objets de règle au lieu d'ID

Lorsque vous utilisez l'expression A.6.17.12, «`create`», à la page 182 pour créer un objet de règle interne, vous pouvez transmettre les données au nouvel objet de règle à l'aide du bloc d'initialisation et/ou d'éléments `specify`.

Si les données que vous transmettez incluent une référence à des données externes qui possède un identificateur (par ex., un dossier identifié par un élément `caseID`), pensez à concevoir la classe de règles de l'objet règle de créé de façon à ce qu'il soit initialisé avec un objet de règle représentant ces données, et non avec une valeur d'ID.

L'utilisation d'un tel objet de règle pour l'initialisation peut augmenter la "sécurité type" de vos données et peut contribuer à empêcher un autre concepteur de règles de créer ses propres objets de règle internes pour la même classe de règles, mais en transmettant accidentellement un ID qui représente un type de données externes différent.

Il est probable que la transmission incorrecte de l'ID entraînera l'échec de règles au moment de l'*exécution* (par ex., car une tentative de conversion d'un objet de règle pour cet ID ne trouvera pas les données sous-jacentes). Par ailleurs, la transmission d'un objet de règle pour sécurité de type permettra au valideur de jeu de règles CER de détecter le problème au moment de la *conception*.

8.12 Développement de méthodes statiques

CER prend en charge diverses expressions susceptibles de fournir les calculs dont vous avez besoin.

Si un calcul métier ne peut pas être implémenté à l'aide d'expressions CER, CER prend en charge l'expression `call` pour vous permettre d'effectuer des appels de votre jeu de règles vers une méthode statique sur une classe Java personnalisée. L'éditeur CER fournit quelques éléments de règles (par exemple, "call" pour permettre aux utilisateurs de définir une méthode statique sur une classe Java personnalisée). Voir l'élément de règle "call" dans 7.4.10, «Call», à la page 126.

Voici un exemple de jeu de règles qui appelle une méthode Java :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_StaticMethodDevelopment"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Income">

    <Attribute name="paymentReceivedDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="mostRecentIncome">
      <type>
        <ruleclass name="Income"/>
      </type>
      <derivation>
        <!-- Au départ, la méthode
            identifyMostRecentIncome_StronglyTyped doit être utilisée.

            En pratique, vous ne devez pas conserver deux versions de
            chaque méthode. Affaiblissez simplement les types d'argument et
            et de retour et conservez une seule méthode.
            -->

        <call class="curam.creole.example.BestPracticeDevelopment"
          method="identifyMostRecentIncome_WeaklyTyped">
          <type>
            <ruleclass name="Income"/>
          </type>
          <arguments>
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>

```

Lorsque vous développez votre méthode statique, vous pouvez commencer par utiliser les classes java générées par CER comme types d'argument et/ou types de retour pour la méthode :

```
package curam.creole.example;

import java.util.List;

import curam.creole.execution.session.Session;
import curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Income;
import curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Person;

public class BestPracticeDevelopment {

    /**
     * Identifie le revenu le plus récent d'une personne.
     *
     * Notez que ce calcul peut être effectué à l'aide d'expressions CER
     * , mais s'affiche ici dans Java uniquement dans le but d'illustrer
     * l'utilisation des types générés lors du développement de méthodes Java statiques à utiliser avec
     * CER.
     *
     * Cette méthode n'est adaptée qu'au développement ; pour
     * la production, voir
     * {@linkplain #identifyMostRecentIncome_WeaklyTyped} ci-dessous.
     *
     * En pratique, vous ne devez pas conserver deux versions de chaque
     * méthode. Affaiblissez simplement les types d'arguments et de retour
     * et conservez une seule méthode.
     */
    public static Income identifyMostRecentIncome_StronglyTyped(
        final Session session, final Person person) {

        Income mostRecentIncome = null;
        final List<? extends Income> incomes =
            person.incomes().getValue();
        for (final Income current : incomes) {
            if (mostRecentIncome == null
                || mostRecentIncome.paymentReceivedDate().getValue()
                    .before(current.paymentReceivedDate().getValue())) {
                mostRecentIncome = current;
            }
        }

        return mostRecentIncome;
    }
}
```

Une fois que le fonctionnement de la méthode Java vous satisfait, vous devez affaiblir les types pour utiliser les objets règles dynamiques au lieu des classes Javagénérées (qui ne doivent pas être utilisées dans un environnement de production) :

```

/**
 * Identifie le revenu le plus récent d'une personne.
 *
 * Notez que ce calcul peut être effectué à l'aide
 * d'expressions CER, mais s'affiche ici dans Java uniquement dans le but d'illustrer
 * l'utilisation des types générés lors du développement de méthodes Java à utiliser
 * avec CER.
 *
 * Cette méthode n'est adapt& qu'à la production ; Pour le développement
 * initial, voir
 * {@linkplain #identifyMostRecentIncome_StronglyTyped} ci-dessus.
 *
 * En pratique, vous ne devez pas conserver deux versions de chaque
 * méthode. Affaiblissez simplement l'argument et retournez les types,
 * puis conservez une seule méthode.
 */
public static RuleObject identifyMostRecentIncome_WeaklyTyped(
    final Session session, final RuleObject person) {

    RuleObject mostRecentIncome = null;
    final List<? extends RuleObject> incomes =
        (List<? extends RuleObject>) person.getAttributeValue(
            "incomes").getValue();
    for (final RuleObject current : incomes) {
        if (mostRecentIncome == null
            || ((Date) mostRecentIncome.getAttributeValue(
                "paymentReceivedDate").getValue())
                .before((Date) current.getAttributeValue(
                    "paymentReceivedDate").getValue())) {
            mostRecentIncome = current;
        }
    }

    return mostRecentIncome;
}

```

ATTENTION :

Si vous modifiez l'implémentation d'une méthode statique, CER ne saura *pas* recalculer automatiquement les valeurs d'attribut qui ont été calculées à l'aide de l'ancienne version de votre méthode statique.

Une fois qu'une méthode statique a été utilisée dans un environnement de production pour les valeurs d'attribut enregistrées, plutôt que de modifier l'implémentation, vous devez créer une nouvelle méthode statique (avec la nouvelle implémentation requise) et modifier vos jeux de règles pour utiliser la nouvelle méthode statique. Lorsque vous publiez vos changements de jeux de règles pour désigner la nouvelle méthode statique, CER recalcule automatiquement toutes les instances des valeurs d'attribut affectées.

Toute méthode statique (ou méthode de propriété) appelée à partir de CER :

- doit *uniquement* dépendre des données qui lui ont été transmises (par ex., son comportement doit être déterministe, en fonction de ses paramètres d'entrée). Elle ne doit pas obtenir les données à partir d'autres sources telles que la base de données, des variables d'environnement ou des globales variables, car cela signifie que CER est incapable de détecter quand ces données ont été modifiées. Dans ce cas, les recalculs ne sont plus fiables ; et
- ne doit *pas* entraîner de dommages collatéraux (tels que l'écriture de données dans la base de données), car CER n'apporte aucune garantie relative à l'ordre dans lequel le traitement a lieu, ni la fréquence à laquelle les méthodes statiques/de propriété seront appelées.

8.13 Eviter les pièges courants lors des tests

L'écriture de tests JUnit pour vos jeux de règles CER est très simple, mais vous devez connaître un certain nombre de pièges à éviter.

Cette section présente quelques exemples de code de test qui, à première vue, semblent corrects, mais qui ne permettent pas d'obtenir le comportement requis.

8.13.1 Eviter `assertEquals` de JUnit

Les tests JUnit héritent des méthodes d'assertion liées aux tests. En général, vous supposez qu'un résultat *attendu* correspond à un résultat *réel*.

Un piège courant consiste à utiliser `assertEquals` de JUnit et de constater que cela ne fonctionne pas correctement pour les comparaisons numériques (avec de surcroît un message d'erreur quelque peu déroutant).

CER convertit toutes les instances de `Number` en son propre format numérique (à l'aide de `java.math.BigDecimal`) avant traitement, pour éviter toute perte de précision. Cette conversion peut être problématique et non intuitive si vous utilisez la méthode `assertEquals` de JUnit.

CER inclut une méthode de remplacement dans une classe auxiliaire. Utilisez `CREOLETestHelper.assertEquals`, qui permet de comparer correctement n'importe quel type de nombre.

Pour les autres types de données, `CREOLETestHelper.assertEquals` se comporte de manière identique à `assertEquals` ; il est donc généralement recommandé d'utiliser `CREOLETestHelper.assertEquals` tout au long de vos tests, pour éviter toute confusion possible (même dans les cas où ce n'est pas nécessaire techniquement).

```

public void creoleTestHelperNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Ne fonctionne pas - getValue renvoie le gestionnaire numérique de CER,
     * mais 65 est un entier.
     * JUnit génère le message quelque peu déroutant :
     * junit.framework.AssertionFailedError: expected:<65> but
     * was:<65>
     *
     * Use CREOLETestHelper.assertEquals instead.
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement()
        .getValue());

}

```

8.13.2 Ne pas oublier d'utiliser `.getValue()`

Le générateur de codes de test CER crée une interface Java pour chaque classe de règles et une méthode d'accès sur l'interface pour chaque attribut de règle.

Cette méthode d'accès générée retourne un attribut CER `AttributeValue`, et *non* la valeur de l'attribut directement. Pour obtenir la valeur proprement dite, vous devez appeler la méthode `.getValue()` sur l'attribut `AttributeValue`.

Si vous oubliez d'utiliser `.getValue()` dans un test, la compilation de votre test s'effectuera correctement, mais son exécution échouera.

```

public void getValueNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Ne fonctionne pas - ageAtRetirement() est un calculateur, et non une
     * valeur.
     *
     * JUnit indique le message suivant :
     * junit.framework.AssertionFailedError: expected:<65> but
     * was: <Value: 65>
     *
     * N'oubliez pas d'utiliser .getValue() sur chaque calculateur d'attribut !
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement());

}

```

Notez que dans cet exemple, la valeur `AttributeValue` est indiquée par la chaîne "Value: 65", et non par le nombre 65 (valeur que `.getValue()` aurait retournée).

8.13.3 N'oubliez pas de spécifier toutes les valeurs requises par les calculs testés

Dans vos tests, vous devez uniquement spécifier les valeurs accessibles lors de l'exécution des règles.

Toutefois, il peut être facile d'oublier de spécifier une valeur ; dans l'affirmative, lorsque CER tente un calcul, il peut rencontrer un attribut dont la dérivation est <specified>, mais pour laquelle aucune valeur n'a été spécifiée dans votre code de test. CER signalera alors une pile d'erreurs :

```
public void valueNotSpecified() {  
  
    final FlexibleRetirementYear flexibleRetirementYear =  
        FlexibleRetirementYear_Factory.getFactory().newInstance(  
            session);  
  
    /**  
    * Ne fonctionne pas - une valeur requise pour le calcul a été marquée  
    * <specified>, mais aucune valeur n'a été spécifiée pour celui-ci.  
    *  
    * CER va signaler une pile de messages :  
    * <ul>  
    *  
    * <li>Erreur lors du calcul de l'attribut 'ageAtRetirement' de la classe de  
    * règles 'FlexibleRetirementYear' (instance id '1', description  
    * 'Undescribed instance of rule class  
    * 'FlexibleRetirementYear', id '1'). </li>  
    *  
    * <li>Erreur lors du calcul de l'attribut 'retirementCause' de la classe de  
    * règles 'FlexibleRetirementYear' (instance id '1', description  
    * 'Undescribed instance of rule class  
    * 'FlexibleRetirementYear', id '1'). </li>  
    *  
    * <li>La valeur doit être spécifiée avant son utilisation (elle ne peut pas être  
    * calculée).</li>  
    *  
    * </ul>  
    *  
    * N'oubliez pas de spécifier toutes les valeurs requises par les calculs !  
    */  
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear  
        .ageAtRetirement().getValue());  
  
}
```

8.13.4 Ne spécifiez pas la même valeur plusieurs fois

CER vous permet de spécifier une valeur qui serait autrement calculée.

Toutefois, lorsque vous utilisez la stratégie RecalculationsProhibited, CER indique une erreur d'exécution si vous tentez de spécifier la valeur d'un attribut (sur un objet de règle particulier) plusieurs fois ; une fois la valeur spécifiée, elle ne peut pas être modifiée (car cela signifie que les calculs précédemment effectués seront «faux»).

```

public void valueSpecifiedTwice() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Ne fonctionne pas - la valeur d'attribut ne peut pas être spécifiée
     * une seconde fois.
     *
     * CER indique le message : Une valeur ne peut pas être spécifiée,
     * car l'état en cours de ce calculateur est 'SPECIFIED'.
     *
     * Ne spécifiez pas deux fois la même valeur !
     */
    flexibleRetirementYear.retirementCause().specifyValue(
        "Lottery winner");

}

```

8.13.5 Spécification du type correct de valeur pour un attribut

Chaque attribut `AttributeValue` possède une méthode `specifyValue` permettant de spécifier (et non de calculer) la valeur d'attribut. La méthode accepte n'importe quelle valeur¹⁸ Si vous spécifiez le type de valeur incorrect, CER indique une erreur d'exécution :

18. Les lecteurs techniques peuvent se demander pourquoi l'attribut `AttributeValue.specifyValue` n'utilise pas de caractères génériques Java 5 pour limiter le type de valeur qu'il peut recevoir.

Si une classe de règles A étend la classe de règles B, A peut remplacer la dérivation de n'importe quel attribut B. A peut également redéclarer n'importe quel attribut B de façon plus restrictive (par ex., la déclaration A de l'attribut indique que son type est un sous-type de celui déclaré par B).

L'interface Java générée pour A étend l'interface Java générée pour B. Les manipulateurs retournant les calculateurs, et non pas directement le type de valeur, toutes les interfaces doivent utiliser une extension générique de sorte que le compilateur autorise la déclaration A du manipulateur de l'attribut à étendre celle de B. L'extension générique étant utilisée, l'attribut `specifyValue` ne peut pas se limiter à un type et doit par conséquent être déclaré pour recevoir un attribut `Object`.

D'un autre point de vue, si une classe Java C doit être étendue en classe Java D, C peut alors définir un type de retour plus restrictif pour l'une des méthodes `get D`, mais ne peut pas restreindre les méthodes d'accès `set` en un sous-type. C doit implémenter la méthode d'accès `set D` et détecter les valeurs non souhaitées au moment de l'exécution (bien qu'une telle pratique puisse enfreindre le principe de substitution de Liskov).

De plus, lorsque des objets règle de purement dynamiques sont utilisés (par ex., dans une session interprétée), la restriction de temps de compilation des valeurs ne peut pas être utilisée.

Par conséquent, CER utilise sa connaissance des types d'attribut déclaré pour détecter des valeurs incorrectes au moment de l'exécution, et non au moment de la compilation.

```

public void incorrectValueType() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    /**
     * Ne fonctionne pas - retirementCause() attend une chaîne, et non pas un
     * nombre.
     *
     * CER va signaler le message : Tentative de définition de la valeur '123'
     * (de type 'java.lang.Integer') sur l'attribut 'retirementCause'
     * de la classe de règles 'FlexibleRetirementYear' (qui attend
     * 'java.lang.String').
     */
    flexibleRetirementYear.retirementCause().specifyValue(123);

}

```

8.13.6 Création de tous les objets de règle d'une session avant exécution des calculs getValue

Vos tests de jeux de règles peuvent configurer n'importe quel nombre d'objets de règle d'une session CER avant de passer à la vérification des valeurs calculées sur ces objets de règle.

Cependant, une fois les calculs commencés, la stratégie `RecalculationsProhibited` empêche de créer un objet de règle invalidant la valeur des calculs A.6.17.53, «readall», à la page 215 précédemment effectués.

Structurez vos tests de sorte que la création de tous vos objets de règle survienne *avant* les calculs (par ex., avant l'exécution de `getValue`). En pratique, ce n'est pas trop restrictif.

Si votre test tente de créer un nouvel objet de règle dans une session après exécution d'un calcul, la stratégie `RecalculationsProhibited` indique une erreur d'exécution (si les calculs `readall` précédemment effectués sont affectés) :

```

public void newObjectsAddedAfterCalculationsStarted() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Calcul de l'âge de départ à la retraite et test de sa valeur
     */
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear
        .ageAtRetirement().getValue());

    /**
     * Création d'un autre objet de règle.
     */

    /**
     * Peut ne pas fonctionner - les nouveaux objets de règle ajoutés à la session une fois
     * les calculs commencés peuvent invalider les calculs
     * <code>readall</code> précédents.
     *
     * {@linkplain RecalculationsProhibited} peut afficher le
     * message : "Impossible de créer de nouveaux objets de règle pour cette session,
     * car celle-ci a déjà accepté une demande
     * de calcul."
     *
     * Pour éviter ce problème, créez tous vos objets de règle avant
     * de tenter un calcul !
     */
    final FlexibleRetirementYear flexibleRetirementYear2 =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

}

```

avertissement : Si votre jeu de règles ne contient pas *actuellement* d'expressions `readall`, vous pouvez ne pas structurer vos tests de sorte que toute création d'objet règle survienne avant un calcul.

Toutefois, si vous modifiez votre jeu de règles dans l'avenir pour contenir les expressions `readall`, vous aurez besoin de restructurer vos tests à ce stade.

Pour éviter tout retravail, structurez toujours vos tests de sorte que toute création d'objet de règle soit exécutée avant le début des calculs.

Annexe A. Dictionnaire CER XML

A.1 Introduction

Cette annexe décrit les éléments qui composent le langage CER des jeux de règles.

A.2 Jeu de règles

Un jeu de règles définit son *nom* et contient plusieurs éléments `Class` de règles et/ou instructions `Include`. L'élément `RuleSet` peut également contenir des éléments `Annotations`.

La structure XML d'un jeu de règles et de ses éléments est limitée par le schéma `RuleSet.xsd` CER. Ce schéma est construit de manière dynamique, de façon à ce que les extensions du CER puissent fournir des expressions et annotations au schéma.

Voici un exemple de jeu de règles :

- RuleSet
 - Annotations (facultatif)
 - ...
 - ...
 - Include
 - ...
 - Include
 - ...
 - ... more Include statements
 - Class
 - Annotations (facultatif)
 - ...
 - ...
 - Initialization (facultatif)
 - Attribute
 - Annotations (facultatif)
 - ...
 - ...
 - type
 - ...
 - Attribute
 - type
 - ...
 - ... more initialized attributes
 - Attribute
 - Annotations (facultatif)
 - ...
 - ...
 - type
 - ...
 - derivation
 - (expression)
 - Annotations (facultatif)
 - ...
 - ...
 - (sous-expression)
 - ...
 - Attribute
 - type
 - ...
 - derivation
 - ...
 - ... plus d'attributs calculés
- ... plus de classes de règles

A.3 Instruction Include

Vous jugerez peut-être pratique de diviser un jeu de règles important en éléments plus petits pour faciliter le développement parallèle ou la réutilisation. Chaque jeu de règles peut contenir des instructions Include pour "extraire" d'autres classes et jeux de règles. L'élément principal d'un élément inclus doit correspondre à l'un des éléments suivants :

- **Class**
Une classe de règles unique ;
- **RuleSet**
Un jeu de règles complet, qui peut lui-même contenir ses propres instructions Include qui seront traitées de manière récursive.

Différentes types d'instructions Include sont prises en charge :

- **RelativePath**

Inclut un fichier XML avec un chemin relatif au fichier contenant ; ce mécanisme peut s'avérer utile lors du développement autonome du jeu de règles par les développeurs à l'aide d'un environnement de développement basé sur des fichiers ;

- **Classpath**

Inclut un fichier XML situé au niveau de l'emplacement nommé du chemin d'accès aux classes d'exécution; ce mécanisme peut s'avérer utile pour faire référence aux jeux de règles communs qui changent rarement et sont générés dans l'application.

Conseil : Dans un jeu de règles, aucune signification n'est attachée à l'ordre que les instructions Include spécifient. Vous êtes libre de réorganiser les instructions Include d'un jeu de règles dans affecter le comportement de ce dernier.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamssoftware.com/CreoleRulesSchema.xsd">

  <!-- Cette classe de règles est définie directement dans ce jeu de règles -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Inclut un jeu de règles défini dans un autre fichier.

      Lorsqu'ils sont rassemblés dans un jeu de règles unique, les
      noms de toutes les classes de règles doivent être uniques. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Voir 3.13, «Consolidateur de jeu de règles CER», à la page 28 pour savoir comment réduire un jeu de règles qui contient des inclusions RelativePath dans un fichier de jeu de règles unique.

A.4 Classe de règles

Une classe de règles permet de définir le comportement de ses instances d'objet de règle.

Une classe de règles définit son *nom* (qui doit être unique parmi les classes de règles du jeu de règles), indique s'il est *abstrait*, et contient les éléments suivants :

- **Initialisation**

Un bloc d'attributs dont les valeurs doivent être spécifiées dès qu'une instance d'objet de règle de la classe est créée ; et

- **Attribut**

Zéro ou plusieurs instructions Attribute calculées, décrivant chacune une valeur pouvant être calculée par la classe de règles.

Une classe de règles peut être définie dans son propre fichier XML (là où l'élément XML racine est `Class`) et incluse à un jeu de règles parent à l'aide d'une instruction A.3, «Instruction Include», à la page 158.

A.4.1 Attributs initialisés

Le bloc `Initialization` contient une ou plusieurs instructions `Attribute`, chacune spécifiant l'élément type de l'attribut, mais *pas* d'élément `derivation`.

Chaque fois qu'une instance d'objet de règle de la classe est créée (dans les règles à l'aide de l'expression `create` ou via le code Java à l'aide des classes de règles générées ou l'API de règle dynamique), les valeurs de tous les attributs initialisés doivent être spécifiés *dans leur ordre de définition dans la classe de règles*.

avertissement : Dans ce cas, évitez de réorganiser les attributs dans un bloc `Initialization`, à moins que vous ne soyez préparé à mettre à jour également tous les espaces (dans les règles ou le code Java) qui créent des instances d'objet de règle de la classe de règles.

A.4.2 Attributs calculés

Les attributs calculés sont répertoriés directement dans la classe de règles.

Conseil : Dans une classe de règles, l'ordre dans lequel les attributs calculés sont spécifiés n'a pas d'importance. Vous êtes libre de réorganiser les attributs calculés dans une classe de règles sans affecter le comportement de la classe de règles.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_RuleClass"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Initialization>
      <!-- Les attributs initialisés possèdent tous un type,
        mais pas de dérivation.

          Vous ne devez PAS réorganiser de façon arbitraire
          les attributs initialisés. -->
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Number"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- Chaque attribut calculé indique
      un type et une dérivation.

          Vous êtes libre de réorganiser arbitrairement
          les attributs calculés. -->
    <Attribute name="isAdult">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="18"/>
        </compare>
      </derivation>
    </Attribute>

    <Attribute name="isSeniorCitizen">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="65"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

A.5 Attribut

Chaque attribut indique son *nom* (qui doit être unique parmi les attributs de règle définis sur/hérités par la classe de règles) et contient ce qui suit :

Chaque élément Attribute contient ce qui suit :

- **type**
Définit le type de valeur que cet attribut fournit (voir 4.4.1, «Types de données pris en charge», à la page 39) ; et
- **dérivation (attributs calculés uniquement)**
Définit comment l'attribut calcule sa valeur. Chaque élément derivation contient une expression CER unique (voir A.6.17, «Liste alphabétique complète des expressions», à la page 164).

Important : Il existe des expressions de marquage spéciales qui peuvent modifier la sémantique de l'attribut de règle - voir A.6.13, «Marqueurs», à la page 164.

A.6 Expressions

CER prend en charge un large éventail d'expressions. Les expressions sont répertoriées par ordre alphabétique ci-dessous, mais sont également regroupées logiquement ici pour référence (notez que des expressions apparaissent intentionnellement dans plusieurs groupes logiques).

A.6.1 Logique booléenne

- A.6.17.68, «true», à la page 239
- A.6.17.26, «false», à la page 193
- A.6.17.2, «all», à la page 167
- A.6.17.3, «any», à la page 169
- A.6.17.46, «not», à la page 206

A.6.2 Comparaison de valeurs

- A.6.17.18, «equals», à la page 189
- A.6.17.10, «compare», à la page 179
- A.6.17.60, «sort», à la page 225

A.6.3 Constantes

Ces expressions fournissent des valeurs de constante littérale.

- A.6.17.68, «true», à la page 239
- A.6.17.26, «false», à la page 193
- A.6.17.47, «null», à la page 206
- A.6.17.62, «String», à la page 227
- A.6.17.48, «Number», à la page 208
- A.6.17.15, «Date», à la page 186
- A.6.17.8, «Code», à la page 178
- A.6.17.33, «FrequencyPattern», à la page 198

A.6.4 Logique conditionnelle

- A.6.17.7, «choose», à la page 176

A.6.5 Agrégations de listes

Ces expressions agrègent une liste de valeur en une valeur dérivée.

- A.6.17.2, «all», à la page 167
- A.6.17.3, «any», à la page 169
- A.6.17.64, «sum», à la page 230
- A.6.17.44, «min», à la page 204
- A.6.17.41, «max», à la page 202
- A.6.17.11, «concat», à la page 180
- A.6.17.58, «singleitem», à la page 223

Pour d'autres opérations fournies directement à partir de l'interface `java.util.List`, voir Annexe B, «Opérations de liste utiles», à la page 247.

A.6.6 Transformations de listes

Ces expressions transforment une liste pour en créer une nouvelle.

- A.6.17.16, «dynamiclist», à la page 186
- A.6.17.32, «fixedlist», à la page 196
- A.6.17.27, «filter», à la page 193
- A.6.17.38, «joinlists», à la page 200
- A.6.17.56, «removeduplicates», à la page 219
- A.6.17.60, «sort», à la page 225
- A.6.17.63, «sublists», à la page 228

A.6.7 Messages localisables

Ces expressions permettent de créer des messages qui peuvent s'afficher dans la langue/l'environnement local de l'utilisateur.

- A.6.17.11, «concat», à la page 180
- A.6.17.57, «ResourceMessage», à la page 221
- A.6.17.69, «XmlMessage», à la page 239

A.6.8 Calculs numériques

Ces expressions prennent en charge les calculs numériques :

- A.6.17.48, «Number», à la page 208
- A.6.17.4, «arithmétique», à la page 171
- A.6.17.50, «periodlength», à la page 210
- A.6.17.64, «sum», à la page 230
- A.6.17.41, «max», à la page 202
- A.6.17.44, «min», à la page 204

A.6.9 Références

Ces expressions permettent à un calcul de faire référence à un autre élément.

- A.6.17.54, «reference», à la page 217
- A.6.17.13, «current», à la page 184
- A.6.17.65, «this», à la page 232

A.6.10 Création

Cette expression permet de créer un nouvel objet de règle.

- A.6.17.12, «create», à la page 182

A.6.11 Récupération

Cette expression permet la récupération d'objets règle.

- A.6.17.53, «readall», à la page 215

A.6.12 Appels Java

Ces expressions permettent d'appeler le code Java pour effectuer un calcul.

- A.6.17.51, «property», à la page 212
- A.6.17.5, «call», à la page 173

A.6.13 Marqueurs

Ces expressions sont des marqueurs spéciaux (Il ne s'agit pas réellement de calculs).

- A.6.17.1, «abstract», à la page 165
- A.6.17.61, «specified», à la page 227

A.6.14 Chronologies

Ces expressions traitent les chronologies CER.

Pour plus de détails sur les chronologies CER, voir 4.5, «Gestion des données variables au fil du temps», à la page 49 dans ce guide.

- A.6.17.36, «Intervalle», à la page 198
- A.6.17.66, «Timeline», à la page 234
- A.6.17.19, «existencetimeline», à la page 191
- A.6.17.37, «intervalvalue», à la page 199
- A.6.17.67, «timelineoperation», à la page 236

A.6.15 Éligibilité et autorisation de distribution de produit

Ces expressions permettent de disposer de calculs métier spécifiques pour l'éligibilité et l'autorisation des dossiers de distribution de produits.

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules* pour obtenir une description de ces expressions.

- A.6.17.9, «combineSuccessionSets», à la page 179
- A.6.17.39, «legislationChange», à la page 202
- A.6.17.52, «rate», à la page 215

A.6.16 householdcomposition

Voir le guide *CGIS Household Composition Expressions*.

A.6.17 Liste alphabétique complète des expressions

Cette section définit toutes les expressions incluses au CER de l'application.

Les expressions ci-dessous sont répertoriées dans l'ordre alphabétique ; voir les sections précédentes des catégorisations utiles de ces expressions.

Remarque : Certaines expressions sont destinées à des dérivations spécifiques de l'application. Ces expressions sont toujours incluses ici, mais le lecteur est renvoyé vers d'autres guides Cúram qui décrivent ces expressions dans leur contexte métier.

Dans un souci de concision, les exemples de jeux de règles sont affichés sans annotations ; en pratique les jeux de règles enregistrés à l'aide de l'éditeur CER contiennent des annotations pour les informations de diagramme et de description.

A.6.17.1 abstract

Expression de marqueur pour indiquer que la dérivation de l'attribut doit être spécifiée sur des sous-classes concrètes (ou l'une de ses superclasses).

Si des attributs d'une classe de règles sont marqués *abstract*, le valideur du jeu de règles CER insiste sur le fait que la classe proprement dite est marquée *abstract="true"* et empêche l'utilisation de la classe de règles dans les expressions A.6.17.12, «create», à la page 182.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_abstract"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Classe de base pour tous les types de prestation.
    Chaque sous-classe concrète possède son propre
    calcul "name" et "isEligible". -->
  <Class name="Benefit" abstract="true">
    <Initialization>
      <!-- Personne pour laquelle l'éligibilité des prestations
        est déterminée. -->
      <Attribute name="person">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- Nom de ce type de prestation -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

    <!-- Eligibilité ou non de la personne à cette prestation. -->
    <Attribute name="isEligible">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

  </Class>

  <!-- Sous-classe concrète de la prestation.
    Contient les dérivations concrètes des attributs abstraits
    hérités. -->
  <Class name="MedicalBenefit" extends="Benefit">
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Medical Benefit"/>
      </derivation>
    </Attribute>
    <Attribute name="isEligible">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- NB/// Attribut de personne hérité de la prestation
-->
              <reference attribute="isPoor">
                <reference attribute="person"/>
              </reference>
              <reference attribute="isSick">
                <reference attribute="person"/>

```

A.6.17.2 all

Fonctionne sur une liste de valeurs booléennes pour déterminer si toutes les valeurs de liste sont définies sur *true*.

Le calcul s'arrête à la première valeur *faux* rencontrée dans la liste. Si la liste est vide, cette expression retourne la valeur *vrai*.

La liste des valeurs booléennes est généralement fournie par une liste A.6.17.32, «fixedlist», à la page 196 ou une A.6.17.16, «dynamiclist», à la page 186.

Conseil : L'ordre des éléments de la liste ne modifie pas la valeur de cette expression ; en revanche, pour des raisons de performances, vous souhaitez peut-être structurer une A.6.17.32, «fixedlist», à la page 196 de sorte que les valeurs «fail fast» soient plus proches du haut de la liste et que les valeurs plus coûteuses soient plus proches du bas de la liste.

Remarque : Depuis Cúram V6, CER ne produit plus de rapports d'erreurs dans les expressions enfant lorsque l'erreur n'affecte pas le résultat global.

Par exemple, si une liste fixe de trois attributs booléens possède ces valeurs :

- true;
- <erreur lors du calcul>; et
- false

le calcul de la valeur all pour ces valeurs est false, car au moins l'un des éléments l'est (le troisième dans la liste), indépendamment du deuxième élément qui retourne une erreur.

A l'opposé, si une autre liste fixe des trois attributs booléens possède ces valeurs :

- true;
- <erreur lors du calcul> ; et
- vrai

le calcul de la valeur all pour ces valeurs renvoie une erreur signalée par le deuxième élément de la liste, car cette erreur empêche de déterminer si tous les éléments possèdent la valeur true.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_all"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLoneParent">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Exemple d'opération <all> sur <fixedlist> -->
        <!-- Pour être considéré "parent isolé", une personne doit
          être à la fois non mariée et avoir au moins un enfant -->
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Nous savons que la plupart des personnes de notre
base de données
                sont mariées. Nous testons d'abord cette condition.

                Si la valeur isMarried n'est pas
                spécifiée pour une personne et que cette personne n'a
                pas d'enfants, la valeur <all> est définie sur false ;
                dans le cas contraire, elle renvoie une erreur indiquant que
                la valeur isMarried n'a pas été spécifiée.
                -->
              <not>
                <reference attribute="isMarried"/>
              </not>
              <not>
                <property name="isEmpty">
                  <object>
                    <reference attribute="children"/>
                  </object>
                </property>
              </not>
            </members>
          </fixedlist>
        </all>
      </derivation>
    </Attribute>

    <Attribute name="hasNoYoungChildren">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Exemple d'opération <all> sur une liste <dynamiclist>.

                Si l'âge d'un enfant ne peut pas être
                calculé et qu'il y a au moins un enfant de moins de 5 ans,
                la valeur <all> est définie sur false ; dans le cas contraire, elle
                renvoie une erreur en indiquant pourquoi l'âge de l'enfant n'a pas pu
                être calculé.
                -->
        <!-- Vérifier que les enfants ont tous plus de 5 ans
-->
        <all>
          <dynamiclist>
            <list>
              <reference attribute="children"/>
            </list>
            <listitemexpression>
              <compare comparison=">";
              <reference attribute="age">
                <current/>
              </reference>
          </dynamiclist>
        </all>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

A.6.17.3 any

Fonctionne comme une liste de valeurs booléennes pour déterminer si toutes les valeurs de liste sont définies sur *true*.

Le calcul s'arrête à la première valeur *true* rencontrée dans la liste. Si la liste est vide, cette expression renvoie la valeur *false*.

La liste des valeurs booléennes est généralement fournie par une A.6.17.32, «fixedlist», à la page 196 ou A.6.17.16, «dynamiclist», à la page 186.

Conseil : L'ordre des éléments de la liste ne modifie pas la valeur de cette expression ; en revanche, pour des raisons de performances, vous voudrez peut être structurer une A.6.17.32, «fixedlist», à la page 196 de sorte que les valeurs «succeed fast» se trouvent à proximité du haut de la liste et que les valeurs plus coûteuses se trouvent plus proches du bas de la liste.

Remarque : Depuis Cúram V6, CER ne produit plus de rapports d'erreurs dans les expressions enfant lorsque l'erreur n'affecte pas le résultat global.

Par exemple, si une liste fixe de trois attributs booléens possède ces valeurs :

- false ;
- <erreur lors du calcul> ; et
- vrai

le calcul de la valeur any renvoie une valeur *true*, car au moins l'un des éléments l'est (le troisième de la liste), indépendamment du deuxième élément qui renvoie une erreur.

A l'opposé, si une autre liste fixe des trois attributs booléens possède ces valeurs :

- false ;
- <erreur lors du calcul> ; et
- false

le calcul de la valeur any pour ces valeurs renvoie l'erreur rapportée par le deuxième élément de la liste, car cette erreur empêche de déterminer si les éléments possèdent la valeur *true*.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_any"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="qualifiesForFreeTravelPass">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Exemple d'opération <any> sur une liste <fixedlist> -->
        <!-- Pour être éligible à un pass de trajets gratuits, la personne
          doit être âgée, non voyante ou handicapée -->
        <any>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Nous savons que la plupart des personnes de notre
                base de données sont des personnes âgées. Par conséquent, nous
                commençons par tester cette condition.

                Si la valeur isBlind n'est pas
                spécifiée pour une personne et que cette personne
                est handicapée, <any> renvoie la valeur false ;
                dans le cas contraire, elle renvoie une erreur indiquant
                que la valeur isBlind n'a pas été spécifiée.
                -->

              <compare comparison=">=">
                <reference attribute="age"/>
                <Number value="65"/>
              </compare>
              <reference attribute="isBlind"/>
              <reference attribute="isDisabled"/>
            </members>
          </fixedlist>
        </any>
      </derivation>
    </Attribute>

    <Attribute name="qualifiesForChildBenefit">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Exemple d'opération <any> sur une liste <dynamiclist>.

          Si l'âge d'un enfant ne peut pas être
          calculé et qu'il y a au moins un enfant de moins de 16 ans,
          <any> renvoie la valeur true ; dans le cas contraire, elle
          renvoie une erreur indiquant pourquoi l'âge de l'enfant n'a pas pu
          être calculé.

          -->
        <!-- Pour être éligible à la prestation pour enfants, cette personne doit
          avoir un ou plusieurs enfants de moins de 16 ans. -->
        <any>
          <dynamiclist>
            <list>
              <reference attribute="children"/>
            </list>
            <listitemexpression>
              <compare comparison="<=">
                <reference attribute="age">
                  <current/>
                </reference>
                <Number value="16"/>
              </compare>
            </listitemexpression>
          </dynamiclist>
        </any>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

A.6.17.4 arithmétique

Effectue un calcul arithmétique sur deux nombres (numéro de gauche et numéro de droite) et arrondit éventuellement le résultat au nombre de décimales spécifiées.

Les opérations prises en charge sont les suivantes :

- **Addition**
nombre de gauche + nombre de droite ;
- **Soustraction**
nombre de gauche - nombre de droite ;
- **Multiplication**
nombre de gauche * nombre de droite ; et
- **Division**
nombre de gauche / nombre de droite.

Si l'arrondissement est nécessaire, vous devez spécifier ce qui suit :

- le nombre de décimales ; et
- la méthode d'arrondissement lorsque celui-ci est exécuté. Voir `JavaDoc for RoundingMode` pour connaître la liste des méthodes d'arrondissement prises en charge et une explication détaillée de leur comportement.

avertissement : Pour les opérations de division, vous devez généralement fournir une méthode d'arrondissement et un nombre de décimales. Si vous ne le faites pas et qu'un résultat exact ne peut pas être calculé au moment de l'exécution, une erreur d'exécution est signalée.

Le valideur de l'éditeur de règles CER émet un avertissement s'il détecte une opération de division dans votre jeu de règles qui n'effectue pas l'arrondissement spécifié.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_arithmetic"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="ArithmeticExampleRuleClass">

    <!-- 3 + 2 = 5 -->
    <Attribute name="addANumberToAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 3 - 2 = 1 -->
    <Attribute name="subtractANumberFromAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="-">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 3 * 2 = 6 -->
    <Attribute name="multiplyANumberByAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="*">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 3 / 2 = 1.5 -->
    <!-- S'agissant d'une division par 2,
           nous pouvons nous abstenir de l'arrondissement.
           Cependant, un avertissement sera toujours émis par
           le valideur de jeu de règles CER. -->
    <Attribute name="divideANumbersByAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="/">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- (3 + 2) * 4 = 20 -->
    <Attribute name="chainedArithmetic">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="*">
          <arithmetic operation="+">
            <Number value="3"/>
            <Number value="2"/>
          </arithmetic>
        </arithmetic>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

A.6.17.5 call

Appelle une méthode Java statique pour effectuer un calcul complexe.

L'expression `call` déclare ce qui suit :

- **type**
Type de données de la valeur renvoyée (voir 4.4.1, «Types de données pris en charge», à la page 39) ; et
- **arguments (facultatif)**
Liste des valeurs à transmettre comme arguments.

La méthode Java doit se trouver sur une classe qui se trouve sur le chemin de classes au moment de la validation du jeu de règles. Le premier argument de la méthode doit être un objet `Session` et les arguments restants doivent correspondre à ceux spécifiés dans le jeu de règles.

avertissement : Vérifiez qu'un code Java appelé par une expression `call` ne tente *pas* de modifier les valeurs des attributs d'objet de règle.

En général, les jeux de règles CER utilisent des types de données non modifiables, mais il est possible d'utiliser vos propres classes Java modifiables en tant que types de données ; si c'est le cas, il est de votre responsabilité de vérifier qu'aucun code appelé n'entraîne la modification du type de données Java personnalisé, car les calculs précédemment effectués deviendraient alors "faux".

```

package curam.creole.example;

import curam.creole.execution.RuleObject;
import curam.creole.execution.session.Session;

public class Statics {

    /**
     * Calcule la couleur préférée d'une personne.
     *
     * Ce calcul est trop complexe pour les règles et a été codé
     * en java.
     *
     * @param session
     *     Session de la règle
     * @param person
     *     Personne
     * @return la couleur préférée calculée de la personne spécifiée
     */
    public static String calculateFavoriteColor(
        final Session session, final RuleObject person) {

        // Notez que la récupération de la valeur d'attribut doit être
        // transtypée vers le type correct
        final String name =
            (String) person.getAttributeValue("name").getValue();
        final Number age =
            (Number) person.getAttributeValue("age").getValue();

        final String ageString = age.toString();
        // Calculer la couleur préférée de la personne en fonction
        // des chiffres inclus dans leur âge et leur nom
        if (ageString.contains("5") || ageString.contains("7")) {
            return "Blue";
        } else if (name.contains("z")) {
            return "Purple";
        } else {
            return "Green";
        }
    }
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_call"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="favoriteColor">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- Appeler une méthode Java statique
              pour effectuer le calcul -->
        <call class="curam.creole.example.Statics"
              method="calculateFavoriteColor">
          <type>
            <javaclass name="String"/>
          </type>
          <arguments>
            <!-- Transmettre à cette personne
                  en tant qu'argument de la
                  méthode statique -->
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

ATTENTION :

Depuis Cúram V6, CER et le gestionnaire de dépendance prennent en charge le recalcul automatique des valeurs calculées par CER si leurs dépendances changent.

Si vous changez l'implémentation d'une méthode statique, CER et le gestionnaire de dépendance ne sauront *pas* automatiquement comment recalculer les valeurs d'attribut qui ont été calculées via l'ancienne version de votre méthode statique.

Une fois qu'une méthode statique a été utilisée dans un environnement de production pour les valeurs d'attribut enregistrées, plutôt que de modifier l'implémentation, vous devez créer une nouvelle méthode statique (avec la nouvelle implémentation requise) et modifier vos jeux de règles pour utiliser la nouvelle méthode statique. Lorsque vous publiez vos changements de jeu de règles afin qu'ils désignent la nouvelle méthode statique, CER et le gestionnaire de dépendance recalculent automatiquement toutes les instances des valeurs d'attribut affectées.

A.6.17.6 catégorie

Voir le guide CGIS Household Composition Expressions.

A.6.17.7 choose

Permet de choisir une valeur basée sur une condition satisfaite.

L'expression choose contient :

- **type**
spécificateur du type de données (voir 4.4.1, «Types de données pris en charge», à la page 39) indiquant le type de valeur qui sera choisi ;
- **test (facultatif)**
expression qui indique la valeur à tester par rapport à la condition dans chaque expression when à son tour. Si aucune expression test n'est spécifiée, l'élément condition de chaque expression when est testé à son tour pour vérifier s'il renvoie la valeur *true* ;
- **when (1 ou plusieurs)**
chacun contient un élément condition à tester et un élément value à renvoyer si la condition est satisfaite ; et
- **otherwise**
expression contenant un élément value à renvoyer (de sorte qu'une valeur soit toujours sélectionnée).

Les conditions sont évaluées dans l'ordre descendant des expressions when. Le traitement s'arrête à la première condition pour effectuer le test. Les conditions suivantes ne sont pas évaluées.

L'expression choose correspond aux expressions `if / else if /.../ else` typiques de la plupart des langages de programmation. Il peut être utilisé efficacement pour implémenter une table de décision.

Vous pouvez envisager de commander vos conditions afin que les personnes les plus susceptibles de réussir se trouvent à proximité du haut de la liste (pour éviter tout calcul inutile).

avertissement : Pour les conditions simples (par ex, celles qui testent l'égalité par rapport à une valeur unique), vous pouvez généralement réorganiser vos conditions sans affecter le comportement du jeu de règles.

Toutefois, pour les conditions plus complexes (et, par conséquent, en général), vous devez déterminer attentivement si la réorganisation de vos conditions introduira des changements de comportement non souhaités.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_choose"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="ageCategory">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <choose>
          <!-- Il n'existe pas de clause <test> explicite. Cette
            instruction <choose> teste chaque condition pour
            vérifier si elles possèdent la valeur TRUE. -->
          <type>
            <javaclass name="String"/>
          </type>

          <!-- Notez que l'ordre de ces conditions est
            important ; si nous devons permuter les positions des
            tests "Newborn" et "Infant", tous les enfants de
            moins de 5 ans (y compris ceux de moins de 1 an) seront
            identifiés comme enfants (Infant) ; aucun enfant ne sera
            identifié comme nourrisson (Newborn). -->
          <when>
            <condition>
              <compare comparison="<">
                <reference attribute="age"/>
                <Number value="1"/>
              </compare>
            </condition>
            <value>
              <String value="Newborn"/>
            </value>
          </when>
          <when>
            <condition>
              <compare comparison="<">
                <reference attribute="age"/>
                <Number value="5"/>
              </compare>
            </condition>
            <value>
              <String value="Infant"/>
            </value>
          </when>
          <when>
            <condition>
              <compare comparison="<">
                <reference attribute="age"/>
                <Number value="18"/>
              </compare>
            </condition>
            <value>
              <String value="Child"/>
            </value>
          </when>
          <otherwise>
            <value>
              <String value="Adult"/>
            </value>
          </otherwise>
        </choose>
      </derivation>
    </Attribute>
  </Class>

```

A.6.17.8 Code

Valeur constante littérale représentant un code d'une table de code d'application.

L'expression Code indique un nom de table de code et prend un argument unique en indiquant la valeur du code nécessaire à partir de la table.

Remarque : Vous devez indiquer la valeur String du code ; les constantes générées par la table de codes ne peuvent pas être utilisées car CER est une langue complètement dynamique et ne peut pas être dépendante des constructions de temps de génération.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Code"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Représentation booléenne du sexe -->
    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Représentation du sexe sous forme de code -->
    <Attribute name="gender">
      <type>
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <Code table="Gender">
          <choose>
            <type>
              <javaclass name="String"/>
            </type>
            <when>
              <condition>
                <reference attribute="isMale"/>
              </condition>
              <value>
                <!-- utiliser le code "MALE" à partir de la table de codes -->
                <String value="MALE"/>
              </value>
            </when>
            <otherwise>
              <value>
                <!-- utiliser le code "FEMALE" de la table de codes -->
                <String value="FEMALE"/>
              </value>
            </otherwise>
          </choose>
        </Code>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.6.17.9 combineSuccessionSets

Voir le guide Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

A.6.17.10 compare

Compare une valeur de gauche à une valeur de droite, en fonction de la comparaison fournie.

Les comparaisons prises en charge sont les suivantes :

- <
le nombre de gauche "est inférieur" au nombre de droite ;
- <=
le nombre de gauche "est inférieur ou égal" au nombre de droite ;
- >
le nombre de gauche "est supérieur" au nombre de droite ; et
- >=
le nombre de gauche "est supérieur ou égal" au nombre de droite.

les valeurs de gauche et de droite peuvent être tout type d'objet comparable, notamment (mais sans s'y limiter) :

- Number;
- String; and
- curam.util.type.Date.

Remarque : Toutes les instances Number sont converties au format numérique de CER (supporté par java.math.BigDecimal) avant comparaison.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_compare"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="CompareExampleRuleClass">

    <!-- 3 >= 2 - TRUE-->
    <Attribute name="compareTwoNumbers">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <Number value="3"/>
          <Number value="2"/>
        </compare>
      </derivation>
    </Attribute>

    <!-- Nouvel An antérieur à Noël - TRUE -->
    <Attribute name="compareTwoDates">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison="&lt; ">
          <Date value="2007-01-01"/>
          <Date value="2007-12-25"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

A.6.17.11 concat

Crée un message localisable (voir 3.3, «Prise en charge de la localisation», à la page 9) en concaténant une liste de valeurs.

Chaînes concat et leurs valeurs sans espaces, ni texte supplémentaires. Si vous avez besoin de formatage ou de texte localisable complexe, pensez à utiliser plutôt A.6.17.57, «ResourceMessage», à la page 221.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_concat"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="surname">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Identificateur d'une personne, notamment
    le nom, le prénom et la date de naissance. Par exemple,
    John Smith (3 octobre 1970).

    Le nom et le prénom sont des chaînes en texte clair,
    mais la date de naissance sera localisée
    en fonction de l'environnement local de l'utilisateur.
    -->
    <Attribute name="personIdentifiant">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <concat>
          <fixedlist>
            <listof>
              <!-- Nous utilisons Object car la liste
              comporte un mélange d'éléments de chaîne
              et de date. -->
              <javaclass name="Object"/>
            </listof>
            <members>
              <reference attribute="firstName"/>
              <!-- espace séparateur entre les noms -->
              <String value=" "/>
              <reference attribute="surname"/>
              <String value=" ("/>
              <reference attribute="dateOfBirth"/>
              <String value=")"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

A.6.17.12 create

Permet d'obtenir une nouvelle instance de classe de règles dans la mémoire de la session. Toutes les valeurs d'initialisation requises par l'objet de règle doivent être spécifiées en tant qu'éléments enfant de l'expression create.

Depuis Cúram V6, l'élément create peut être utilisé pour créer une instance de classe de règles à partir d'un jeu de règles *différent*, en définissant la valeur de l'attribut XML `ruleset` facultatif.

Remarque : Les objets de règle créés via l'élément create ne peuvent pas être extraits lors de l'exécution des règles, car cela violerait le principe d'organisation de CER.

Depuis Cúram V6, vous avez le choix entre plusieurs syntaxes pour transmettre les valeurs à un objet de règle créé :

- **bloc d'initialisation**

CER continue de prendre en charge un bloc d'attributs définis dans un élément `Initialization`. Cette syntaxe peut être utile pour les attributs qui *doivent* toujours être définis et n'ont pas d'implémentation par défaut ; et

- **spécifier des éléments**

Depuis Cúram V6, CER prend également en charge des attributs arbitraires dont leur valeur est remplacée par un élément `specify` qui nomme l'attribut à définir et contient la valeur à utiliser. Cette syntaxe peut être utile pour les attributs qui ne sont définis que parfois et/ou possèdent une implémentation par défaut.

Depuis Cúram V6, les objets de règle créés sont «regroupés» dans la session. Ce pool permet aux demandes identiques de créer un objet de règle afin qu'il soit traité par un seul objet de règle, qui peut conserver l'utilisation de la mémoire et également empêcher l'exécution de calculs en double pour réduire la charge UC. Deux demandes de création d'un objet de règle sont jugées identiques si elles demandent la même classe de règle et que les valeurs des attributs initialisés et spécifiés sont égales.

Dans l'exemple ci-dessous, si le numéro de téléphone professionnel d'une personne est identique à son numéro de téléphone personnel, un seul objet de règle sera utilisé pour chaque numéro. Par conséquent, la valeur dérivée pour `isOutOfThisArea` ne sera calculée qu'une seule fois. Si les numéros professionnel et personnel sont différents, deux objets de règle seront créés.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_create"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <!-- Détails des numéros de téléphone regroupés dans les informations collectées -->
    <Attribute name="homePhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="homePhoneNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Création d'objets de règle PhoneNumber
    et placement dans une liste -->
    <Attribute name="phoneNumbers">
      <type>
        <javaclass name="List">
          <ruleclass name="PhoneNumber"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="PhoneNumber"/>
          </listof>
        </fixedlist>
      </derivation>
      <members>

        <!-- Détails du numéro de téléphone personnel. -->
        <create ruleclass="PhoneNumber">
          <!-- Valeur de PhoneNumber.owner -->
          <this/>
          <!-- Valeur de PhoneNumber.number -->
          <reference attribute="homePhoneNumber"/>
          <specify attribute="areaCode">
            <!-- Valeur de PhoneNumber.areaCode -->
            <reference attribute="homePhoneAreaCode"/>
          </specify>
        </create>
      </members>
    </Attribute>
  </Class>

```

A.6.17.13 current

Fait référence à un élément dans une liste en cours de traitement.

L'expression `current` peut uniquement apparaître dans une expression qui traite des éléments dans une liste, par exemple :

- `listitemexpression` dans une expression A.6.17.27, «`filter`», à la page 193 ou A.6.17.16, «`dynamiclist`», à la page 186 ; ou
- `sortorder` dans une expression A.6.17.60, «`sort`», à la page 225.

Pour plus de clarté, vous pouvez affecter un alias à l'expression `current`, qui doit correspondre à l'alias de l'expression `list` dont il est fait référence. Les alias sont nécessaires si le même calcul comporte davantage d'expressions `current` dans la portée.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_listitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="adults">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <!-- La référence utilise current pour faire référence
              à un élément présent dans la liste des
              objets de règle Person. -->
            <reference attribute="isAdult">
              <current/>
            </reference>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isAdult">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="18"/>
        </compare>
      </derivation>
    </Attribute>
  </Class>

```

A.6.17.14 currentunitmember

Voir le guide CGIS Household Composition Expressions.

A.6.17.15 Date

Valeur constante de date littérale, de type `curam.util.type.Date`.

La valeur `Date` est indiquée sous la forme `aaaa-mm-jj`.

Remarque : Il n'existe aucune fonction dans CER pour obtenir la date en cours. Une telle fonction serait instable car la valeur renvoyée par une fonction pourrait varier d'un jour à un autre.

Les fonctions instables sont interdites dans CER, car si le résultat d'une fonction peut changer, cela peut signifier que les calculs précédemment effectués peuvent être maintenant "incorrects".

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Date"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="DateExampleRuleClass">

    <Attribute name="nullDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- A null Date -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- Date 3 octobre 1970 -->
        <Date value="1970-10-03"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.6.17.16 dynamiclist

Crée une liste en évaluant une expression sur chaque élément d'une liste existante.

La nouvelle liste aura une entrée correspondante pour chaque entrée de la liste existante, avec l'organisation conservée.

Une expression `dynamiclist` indique ce qui suit :

- **list**
Liste existante ; et
- **listitemexpression**
Expression à évaluer sur chaque élément de la liste existante.

Un élément `dynamiclist` peut être utilisé lorsque le nombre d'éléments de la liste souhaitée n'est pas connu au moment de la conception (par ex., il peut différer entre deux exécutions, selon la valeur des autres attributs). Si le nombre d'éléments est fixe (par exemple, non connu au moment de la conception), pensez à utiliser plutôt A.6.17.32, «`fixedlist`», à la page 196.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_dynamiclist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isDisabled">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="totalIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="containsDisabledPerson">
      <type>

```

A.6.17.17 defaultDescription

Fournit une implémentation par défaut de l'attribut `description` dont toutes les classes de règles héritent de la classe de règle racine. Voir 4.4.1, «Types de données pris en charge», à la page 39.

Chaque classe de règle doit remplacer l'attribut `description` de la classe de règle racine pour fournir une description plus significative. Si aucun remplacement n'est fourni (ou hérité), un avertissement est émis lorsque le jeu de règles est validé.

Important : L'expression `defaultDescription` ne peut être utilisée *que* par la classe de règle racine. Vous ne devez pas l'utiliser dans vos propres classes de règle.

```
<Class name="RootRuleClass" abstract="true">
  <Attribute name="description">
    <type>
      <javaclass name="curam.creole.value.Message"/>
    </type>
    <derivation>
      <!-- A utiliser EXCLUSIVEMENT dans RootRuleClass -->
      <defaultDescription/>
    </derivation>
  </Attribute>
</Class>
```

A.6.17.18 equals

Détermine si deux objets (un objet à gauche et un objet à droite) sont égaux.

Number valeurs sont converties au format numérique CER (supporté par `java.math.BigDecimal`) avant comparaison ; les différences de zéros de début et de fin sont ignorées.

Les valeurs `null` sont comparées en toute sécurité ; si la valeur de gauche et la valeur de droite sont définies sur `null`, l'expression `equals` renvoie la valeur `true` ; si la valeur de gauche ou la valeur de droite est définie sur `null`, l'expression `equals` renvoie la valeur `false`.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_equals"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="EqualsExampleRuleClass">

    <!-- TRUE -->
    <Attribute name="identicalStrings">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <String value="A String"/>
          <String value="A String"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSE -->
    <Attribute name="differentStrings">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <String value="A String"/>
          <String value="A different String"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- TRUE -->
    <Attribute name="identicalNumbers">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <!-- Ces nombres sont identiques,
            sans tenir compte des
            différences entre les zéros de début et de
            fin -->
          <Number value="123"/>
          <Number value="000123.000"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSE -->
    <Attribute name="differentTypes">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <!-- Ces objets sont de type
            différent ; ils ne sont donc pas
            égaux, même s'ils
            "semblent" identiques.-->
          <String value="123"/>
          <Number value="123"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSE -->
    <Attribute name="oneNull">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>

```

A.6.17.19 existencetimeline

Crée une chronologie de type spécifié à partir d'une paire de dates de début et de fin, dont l'une est facultative.

Voir 4.5.3, «Construction de chronologies», à la page 57.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_existencetimeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- null si la personne est toujours en vie -->
    <Attribute name="dateOfDeath">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Crée une chronologie false avant la naissance
      de la personne, true lorsque la personne est vivante et false après le
      décès de la personne. Si la personne n'a pas de date de décès,
      il n'y a pas d'intervalle "false" de fin. -->
    <Attribute name="isAliveTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <existencetimeline>
          <intervaltype>
            <javaclass name="Boolean"/>
          </intervaltype>
          <intervalfromdate>
            <reference attribute="dateOfBirth"/>
          </intervalfromdate>
          <intervaltodate>
            <reference attribute="dateOfDeath"/>
          </intervaltodate>
          <preExistenceValue>
            <false/>
          </preExistenceValue>
          <existenceValue>
            <true/>
          </existenceValue>
          <postExistenceValue>
            <false/>
          </postExistenceValue>
        </existencetimeline>

        </derivation>
      </Attribute>

    <!-- Crée une chronologie "Before Birth" avant la naissance
      de la personne, "During Lifetime" lorsque la personne est vivante et
      "After Death" lorsque la personne est décédée. Si la personne n'a pas
      de date de décès, l'intervalle "After
      Death" n'existe pas. -->
    <Attribute name="lifeStatus">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="String"/>
        </javaclass>

```

A.6.17.20 facategory

Voir le guide CGIS Household Composition Expressions.

A.6.17.21 faexception

Voir le guide CGIS Household Composition Expressions.

A.6.17.22 faexceptionothermembers

Voir le guide CGIS Household Composition Expressions.

A.6.17.23 fahouseholdcomposition

Voir le guide CGIS Household Composition Expressions.

A.6.17.24 faoptionalmembers

Voir le guide CGIS Household Composition Expressions.

A.6.17.25 farelationship

Voir le guide CGIS Household Composition Expressions.

A.6.17.26 false

Valeur de constante booléenne «false».

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_false"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FalseExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <false/>
        </not>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <false/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.6.17.27 filter

Crée une liste contenant tous les éléments d'une liste existante qui correspondent à la condition de filtre.

L'expression filter contient ce qui suit :

- **list**
liste existante à filtrer ; et
- **listitemexpression**

test à appliquer à chaque élément de la liste.

Généralement, `listitemexpression` contient un ou plusieurs calculs appliqués à l'élément A.6.17.13, «current», à la page 184 dans la liste.

L'ordre relatif des éléments de liste dans le résultat filtré préservera l'ordre des éléments de liste de la liste d'origine. Si aucun élément de la liste ne correspond à la condition de filtre, une liste vide est renvoyée.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_filter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- L'épouse de cette personne, ou
    null si non mariée -->
    <Attribute name="spouse">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Enfants de cette personne -->
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Household">

    <!-- Toutes les personnes du ménage -->
    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Tous les adultes du ménage -->
    <Attribute name="adultMembers">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <compare comparison=">=">

```

A.6.17.28 filtercategory

Voir le guide CGIS Household Composition Expressions.

A.6.17.29 financialunitcategory

Voir le guide CGIS Household Composition Expressions.

A.6.17.30 financialunitdetermination

Voir le guide CGIS Household Composition Expressions.

A.6.17.31 financialunitmember

Voir le guide CGIS Household Composition Expressions.

A.6.17.32 fixedlist

Crée une nouvelle liste à partir d'éléments connus au moment de la conception.

L'expression `fixedlist` indique ce qui suit :

- **listof**
Type d'élément dans la liste renvoyée (voir 4.4.1, «Types de données pris en charge», à la page 39) ; et
- **members**
Eléments dans la liste.

La liste créée contient ses membres dans l'ordre répertoriée dans le jeu de règles.

Conseil : L'élément `members` peut contenir 0, 1 ou plusieurs éléments enfants.

Toutefois, si `fixedlist` se trouve dans une opération de traitement de liste mais ne spécifie que 0 ou 1 membre de liste, le valideur de jeu de règles CER émet un avertissement indiquant que la liste peut être inutile.

Si vous devez créer une liste où le nombre d'éléments de la liste n'est pas connu au moment de la conception, pensez à utiliser plutôt A.6.17.16, «`dynamiclist`», à la page 186.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_fixedlist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Animaux appartenant à cette personne -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>

        <!-- Liste fixe d'animaux -->
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Chaque personne possède exactement deux animaux,
              Skippy et Lassie -->
            <create ruleclass="Pet">
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
              <String value="Lassie"/>
              <String value="Dog"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <Attribute name="isEntitledToBenefits">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <!-- Liste fixe de conditions booléennes -->
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Doit être un adulte -->
              <compare comparison=">=">
                <reference attribute="age"/>
                <Number value="18"/>
              </compare>
              <!-- Doit être résident dans l'état -->
              <reference attribute="isResidentInTheState"/>
              <!-- Doit déclarer des revenus inférieurs au seuil des prestations
-->
              <compare comparison="< ">
                <reference attribute="totalIncome"/>
                <Number value="100"/>
              </compare>
            </members>
          </fixedlist>
        </all>

      </derivation>
    </Attribute>

    <Attribute name="totalIncome">
      <type>

```

A.6.17.33 FrequencyPattern

Valeur constante FrequencyPattern littérale, de type curam.util.type.FrequencyPattern.

La valeur FrequencyPattern est un nombre à 9 chiffres. Voir le JavaDoc pour curam.util.type.FrequencyPattern pour connaître la signification de la chaîne numérique.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_FrequencyPattern"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FrequencyPatternExampleRuleClass">

    <Attribute name="nullFrequencyPattern">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
      <derivation>
        <!-- FrequencyPattern null -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="weeklyOnMondays">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
      <derivation>
        <!-- La chaîne de modèle de fréquence pour
              "Weekly on Mondays" -->
        <FrequencyPattern value="100100100"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.6.17.34 HCCurrent

Voir le guide CGIS Household Composition Expressions.

A.6.17.35 householdcomposition

Voir le guide CGIS Household Composition Expressions.

A.6.17.36 Intervalle

Crée un intervalle (voir 4.5, «Gestion des données variables au fil du temps», à la page 49) de type donné, avec une valeur valide à partir d'une date spécifiée.

Cette expression est généralement utilisée dans le cadre de la construction d'une A.6.17.66, «Timeline», à la page 234.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Interval"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateInterval">

    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- Un autre intervalle-->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <!-- Crée un intervalle du type spécifié.
                  Généralement utilisé comme entrée dans une <chronologie>. -->
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>

            </intervals>
          </Timeline>

        </derivation>
      </Attribute>

    </Class>
  </RuleSet>

```

A.6.17.37 intervalvalue

Encapsule une expression qui renvoie une chronologie(voir 4.5, «Gestion des données variables au fil du temps», à la page 49), et permet à une expression contenant d'être exécutée sur des valeurs individuelles de la chronologie. Cette expression empêche efficacement une expression externe de savoir si elle est exécutée sur une chronologie.

Cette expression ne peut être utilisée que lorsqu'elle est imbriquée dans une expression A.6.17.67, «`timelineoperation`», à la page 236. Pour une description plus détaillée et des exemples d'utilisation de `intervalvalue`, voir A.6.17.67, «`timelineoperation`», à la page 236.

A.6.17.38 `joinlists`

Permet de créer une liste en regroupant des listes existantes.

L'expression `joinlists` prend un argument simple qui doit être une liste de listes.

L'ordre des éléments dans la nouvelle liste est identique à l'ordre dans leur liste source. Les listes sont regroupées dans l'ordre fourni.

Si les listes regroupées peuvent contenir des éléments en double, pensez à encapsuler l'expression `joinlists` dans une expression A.6.17.56, «`removeduplicates`», à la page 219.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_joinlists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Obtention de tous les animaux du ménage,
      en regroupant la liste d'animaux
      de chaque personne -->
    <Attribute name="allPets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <joinlists>
          <!-- liste de liste d'animaux, une
            liste par membre du
            ménage -->
          <dynamiclist>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <reference attribute="pets">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>

          </joinlists>
        </derivation>
      </Attribute>

```

A.6.17.39 legislationChange

Voir le guide Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

A.6.17.40 mandatorymembers

Voir le guide CGIS Household Composition Expressions.

A.6.17.41 max

Détermine la plus grande valeur dans une liste (ou null si la liste est vide).

La liste peut contenir n'importe quel type d'objet comparable, y compris (mais sans s'y limiter) :

- Number ;
- String ; et
- curam.util.type.Date.

Remarque : Toutes les instances Number sont converties au propre format numérique de CER (supporté par `java.math.BigDecimal`) avant comparaison.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_max"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MaxExampleRuleClass">

    <!-- Sélection de "Cherry" comme valeur de chaîne "largest" -->
    <Attribute name="alphabeticallyLastFruit">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <max>
          <reference attribute="fruits"/>
        </max>
      </derivation>
    </Attribute>

    <Attribute name="fruits">
      <type>
        <javaclass name="List">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <javaclass name="String"/>
          </listof>
          <members>
            <String value="Apple"/>
            <String value="Banana"/>
            <String value="Cherry"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Détermine le nombre de taches que porte le chien le plus tacheté -->
    <Attribute name="largestNumberOfSpots">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <max>
          <dynamiclist>
            <list>
              <reference attribute="dalmatians"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfSpots">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </max>
      </derivation>
    </Attribute>

    <Attribute name="dalmatians">
      <type>
        <javaclass name="List">
          <ruleclass name="Dalmation"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

```

A.6.17.42 mealgroupmembers

Voir le guide CGIS Household Composition Expressions.

A.6.17.43 membersforhohunit

Voir le guide CGIS Household Composition Expressions.

A.6.17.44 min

Détermine la plus petite valeur dans une liste (null si la liste est vide).

La liste peut contenir n'importe quel type d'objet comparable, y compris (mais sans s'y limiter) :

- Number;
- String ; et
- curam.util.type.Date.

Remarque : Toutes les instances Number sont converties au format numérique de CER (supporté par java.math.BigDecimal) avant comparaison.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_min"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MinExampleRuleClass">

    <!-- Sélection de New Year comme valeur de date "earliest" -->
    <Attribute name="earliestDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <min>
          <reference attribute="publicHolidays"/>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="publicHolidays">
      <type>
        <javaclass name="List">
          <javaclass name="curam.util.type.Date"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <javaclass name="curam.util.type.Date"/>
          </listof>
          <members>
            <Date value="2007-01-01"/>
            <Date value="2007-12-25"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Détermine le nombre de rayures que comporte le zèbre qui en a le
moins-->
    <Attribute name="smallestNumberOfStripes">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <min>
          <dynamiclist>
            <list>
              <reference attribute="zebras"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfStripes">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="zebras">
      <type>
        <javaclass name="List">
          <ruleclass name="Zebra"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

```

A.6.17.45 multipersonhousehold

Voir le guide CGIS Household Composition Expressions.

A.6.17.46 not

Négation d'une valeur booléenne.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_not"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLivingInUSA">

      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Notez que cela est quelque peu artificiel.
-->
        <not>
          <reference attribute="isLivingOutsideUSA"/>
        </not>
      </derivation>
    </Attribute>

    <Attribute name="isLivingOutsideUSA">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <equals>
            <reference attribute="country"/>
            <String value="USA"/>
          </equals>
        </not>
      </derivation>
    </Attribute>

    <!-- Pays dans lequel réside cette personne. -->
    <Attribute name="country">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.6.17.47 null

Valeur de constante null.

La définition d'une valeur sur null peut être utile pour indiquer qu'aucune valeur n'est appliquée.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_null"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>
  </Class>
  <Class name="Person">
    <!-- Animal préféré de cette personne ou
      null si la personne n'a pas d'animal. -->
    <Attribute name="favoritePet">
      <type>
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
    <!-- Le nom de l'animal préféré
      de cette personne ou null si
      la personne n'a pas d'animal.

      Nous devons tester si favoritePet
      est null avant d'exécuter le
      calcul (simple).-->
    <Attribute name="favoritePetsName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <choose>
          <type>
            <javaclass name="String"/>
          </type>
          <when>
            <!-- Si cette personne n'a pas
              d'animal préféré, définissez le
              nom de l'animal préféré sur null. -->
            <condition>
              <equals>
                <reference attribute="favoritePet"/>
                <null/>
              </equals>
            </condition>
            <value>
              <null/>
            </value>
          </when>
          <otherwise>
            <value>
              <!-- Obtention du nom de l'animal préféré -->
              <reference attribute="name">
                <reference attribute="favoritePet"/>
              </reference>
            </value>
          </otherwise>
        </choose>
      </derivation>
    </Attribute>
  </Class>

```

A.6.17.48 Number

Valeur constante numérique littérale.

Un nombre dans CER est une valeur décimale arbitrairement longue, spécifiée par un point («.») comme séparateur décimal et sans séparateur de milliers.

Les calculs métier CER peuvent souvent impliquer des valeurs en pourcentage (par ex., "Déduire 10% des revenus de la personne". Pour faciliter la codification de ces règles, CER permet de spécifier un nombre comme pourcentage, en ajoutant simplement au nombre le suffixe %. Par exemple, les nombres 12.345% et 0.12345 se comportent de manière identique dans les calculs (mais la version de pourcentage s'affiche sous forme de pourcentage).

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Number"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="NumberExampleRuleClass">

    <Attribute name="aPositiveInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Entier positif -->
        <Number value="1"/>
      </derivation>
    </Attribute>

    <Attribute name="aNegativeInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Entier négatif -->
        <Number value="-2"/>
      </derivation>
    </Attribute>

    <Attribute name="aDecimalNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Nombre décimal.

          Les nombres sont arbitrairement longs/précis ; utilisez "." comme
          séparateur décimal. Les nombres n'ont pas de séparateur
          de milliers.

          -->
        <Number value="-12345.6789"/>
      </derivation>
    </Attribute>

    <Attribute name="aPercentage">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Pourcentage
          (12.345% est équivalent au nombre 0.12345) -->
        <Number value="12.345%"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

A.6.17.49 optionalmembers

Voir le guide CGIS Household Composition Expressions.

A.6.17.50 periodlength

Calcule la quantité d'unités temporelles entre deux dates.

L'une des unités de temps suivantes doit être spécifiée :

- *jours*;
- *semaines*;
- *moins* ; et
- *années*.

L'expression *periodlength* doit également indiquer si la date de fin de la période est *inclusive* ou *exclusive* ou la date de fin (la période est toujours inclusive de la date de début).

Le calcul de la période est toujours arrondi à l'entier le plus proche.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_periodlength"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="PeriodLengthExampleClass">

    <!-- NB 1970 n'était pas une année bissextile -->
    <Attribute name="firstDayOfJanuary1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="lastDayOfDecember1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-12-31"/>
      </derivation>
    </Attribute>

    <Attribute name="firstDayOfJanuary1971">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1971-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="sameDay_LengthInDays">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- démarre et se termine sur le même jour = 1 jour -->
        <periodlength endDateInclusion="inclusive" unit="days">
          <reference attribute="firstDayOfJanuary1970"/>
          <reference attribute="firstDayOfJanuary1970"/>
        </periodlength>
      </derivation>
    </Attribute>

    <Attribute name="sameDay_LengthInWeeks">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- démarre et se termine sur le même jour = 0 semaine-->
        <periodlength endDateInclusion="exclusive" unit="weeks">
          <reference attribute="firstDayOfJanuary1970"/>
          <reference attribute="firstDayOfJanuary1970"/>
        </periodlength>
      </derivation>
    </Attribute>

    <Attribute name="januaryToDecember_LengthInDays">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- 365 jours -->
        <periodlength endDateInclusion="inclusive" unit="days">
          <reference attribute="firstDayOfJanuary1970"/>
          <reference attribute="lastDayOfDecember1970"/>
        </periodlength>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

A.6.17.51 property

Permet d'obtenir la propriété d'un objet Java.

L'expression `property` indique le nom de la méthode Java à appeler, et :

- **object**
Objet Java sur lequel intervenir ; et
- **arguments**
Le cas échéant, une liste d'arguments à transmettre à la méthode Java.

L'expression `property` permet à CER de tirer parti de la puissance des classes Java sans avoir à répliquer un sous-ensemble arbitraire de méthodes en tant qu'expressions CER. Par exemple, `java.util.List` contient une méthode `size` et donc CER ne contient aucune expression explicite pour calculer le nombre d'éléments dans une liste.

Toutefois, pour respecter le principe de l'immuabilité de CER, seules les méthodes Java qui ne modifient pas la "valeur" d'un objet peuvent être appelées. CER permet uniquement d'appeler une méthode "property" si la méthode est incluse dans la "liste sécurisée" des méthodes pour la classe de l'objet (ou l'une de ses classes ou interface ancêtre).

Une méthode est jugée sûre si elle est explicitement indiquée comme telle dans la liste sécurisée. Si elle n'est pas présente dans la liste de sécurité, la règle CER émet une erreur.

Conseil : Le paramètre explicite de la sécurité défini sur *false* n'est pas nécessaire mais peut être inclus pour l'exhaustivité de la documentation, comme c'est le cas avec les listes sécurisées incluses à CER.

La liste sécurisée d'une classe est un fichier de propriétés dans le même package que la classe, nommé `<classname>_CREOLE.properties`.

CER inclut des listes sécurisées pour les classes et les interfaces Java suivantes :

- `java.lang.Object`;
- `java.lang.Number`; and
- `java.util.List`.

```
# Safe list for java.lang.Object

# safe
toString.safe=true

# force equality to be evaluated using <equals>
equals.safe=false

# not exposed, even though they're "safe"
hashCode.safe=false
getClass.safe=false
```

Figure 35. Liste sécurisée pour méthodes `java.lang.Object`

```
# Safe list for java.lang.Number  
  
byteValue.safe=true  
doubleValue.safe=true  
floatValue.safe=true  
intValue.safe=true  
longValue.safe=true  
shortValue.safe=true
```

Figure 36. Safe list for java.lang.Number methods

```
# Safe list for java.util.List  
  
contains.safe=true  
containsAll.safe=true  
  
get.safe=true  
  
indexOf.safe=true  
isEmpty.safe=true  
lastIndexOf.safe=true  
size.safe=true  
subList.safe=true  
  
# not exposed  
hashCode.safe=false  
listIterator.safe=false  
iterator.safe=false  
toArray.safe=false  
  
# mutators - unsafe  
add.safe=false  
addAll.safe=false  
clear.safe=false  
remove.safe=false  
removeAll.safe=false  
retainAll.safe=false
```

Figure 37. Safe list for java.util.List methods

Pour obtenir une description des propriétés utiles sur l'interface List Java, voir Annexe B, «Opérations de liste utiles», à la page 247.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_property"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Indique si cette personne a des enfants.

      Teste la propriété isEmpty de la liste. -->
    <Attribute name="hasChildren">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <property name="isEmpty">
            <object>
              <reference attribute="children"/>
            </object>
          </property>
        </not>

        </derivation>
      </Attribute>

    <!-- Tous les enfants de cette personne, à l'exception du premier.

      Utilise la propriété subList de la liste, en transmettant :
      - (inclusif) de l'élément en position "1" (indiquant le
deuxième
      membre de la liste ; les listes Java sont à base zéro)
      - (exclusif) à l'élément en position "taille de liste" (indiquant
      la position après le dernier élément de la liste)
    -->
    <Attribute name="secondAndSubsequentChildren">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <property name="subList">
          <object>
            <reference attribute="children"/>
          </object>
          <arguments>
            <!-- Le nombre doit être converti en entier
              (comme requis par List.subList). -->
            <property name="intValue">
              <object>
                <Number value="1"/>
              </object>
            </property>
            <property name="size">
              <object>
                <reference attribute="children"/>
              </object>
            </property>
          </arguments>
        </property>
      </derivation>
    </Attribute>
  </Class>

```

ATTENTION :

Depuis Cúram V6, CER et le gestionnaire de dépendance prend en charge le stockage des valeurs d'attribut calculées sur la base de données, ainsi que le recalcul automatique des valeurs d'attribut si leurs dépendances changent.

Si vous changez l'implémentation d'une méthode de propriété, CER et le gestionnaire de dépendance ne sauront *pas* automatiquement comment recalculer les valeurs d'attribut calculées à l'aide de l'ancienne version de votre méthode `property`.

Une fois qu'une méthode `property` a été utilisée dans un environnement de production pour les valeurs d'attribut enregistrées, plutôt que de changer l'implémentation, pensez à créer une nouvelle méthode `property` (avec la nouvelle implémentation requise) et changez vos jeux de règles pour utiliser la nouvelle méthode `property`. Lorsque vous publiez vos changements de jeux de règles afin qu'ils désignent la nouvelle méthode `property`, CER recalcule automatiquement toutes les instances des valeurs d'attribut affectées.

A.6.17.52 `rate`

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

A.6.17.53 `readall`

Extrait toutes les instances d'objet de règle d'une classe de règles (par ex., toutes celles créées par le code client). Les instances d'objet de règle interne (par ex., celles créées à partir des règles) ne sont *pas* extraites.

Voir 4.3, «Objets de règle externes et internes», à la page 33 pour plus d'informations sur la création d'objets de règle.

Depuis Cúram V6, l'élément `readall` peut être utilisé pour extraire les instances d'une classe de règles à partir d'un jeu de règles *différent*, en définissant la valeur de l'attribut XML `ruleset` facultatif.

Depuis Cúram V6, l'expression `readall` prend en charge un élément `match` facultatif qui limite l'activité de l'expression `readall` à l'unique extraction des objets de règle dont la valeur d'un attribut particulier correspond à celle incluse dans le critère de recherche.

Important : Avant Cúram V6, un moyen de récupérer des objets de règle qui correspondaient à un critère consistait à encapsuler un élément `readall` dans une expression A.6.17.27, «`filter`», à la page 193.

Toutefois, pour les sessions CER qui utilisent `DatabaseDataStorage` (voir 4.2, «Sessions CER», à la page 31), il est en général plus efficace d'utiliser la syntaxe `readall / match` introduite dans Cúram V6. La nouvelle syntaxe est plus efficace :

- lorsque l'attribut contenant l'expression `readall` est calculée en premier ; et
- lorsque CER et le gestionnaire de dépendance identifient que l'attribut contenant l'expression `readall` est dépassée et doit être recalculée (voir Chapitre 5, «Gestionnaire de dépendance», à la page 77).

Dans les situations où les objets de règle doivent correspondre à plusieurs critères, utilisez la syntaxe `readall / match` pour baser la correspondance sur l'attribut le plus sélectif, puis encapsulez les résultats dans un A.6.17.27, «`filter`», à la page 193 pour filtrer les autres critères.

Conseil : Si vous souhaitez qu'il s'agisse uniquement d'une instance de singleton de la classe de règles (peut-être après filtrage ou correspondance), pensez à encapsuler l'expression dans une expression A.6.17.58, «`singleitem`», à la page 223.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_readall"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="socialSecurityNumber">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
    Extraction de la seule réclamation qui aura été utilisée pour
    alimenter la session
    -->

    <Attribute name="claim">
      <type>
        <ruleclass name="Claim"/>
      </type>
      <derivation>
        <singleitem onEmpty="error" onMultiple="error">
          <readall ruleclass="Claim"/>
        </singleitem>
      </derivation>
    </Attribute>

    <!--
    Extraction des objets de règle de prestation pour cette personne (créés à partir
    du code client, probablement en interrogeant la mémoire externe).

    Cette implémentation utilise une expression <readall> avec un élément <match> imbriqué pour
    extraire uniquement les objets de règle correspondants et (selon le stockage
    des données) sera plus performante que
    l'implémentation "benefitsFilterReadall" ci-dessous.
    -->

    <Attribute name="benefitsReadallMatch">
      <type>
        <javaclass name="List">
          <ruleclass name="Benefit"/>
        </javaclass>
      </type>
      <derivation>
        <readall ruleclass="Benefit">
          <match retrievedattribute="socialSecurityNumber">
            <reference attribute="socialSecurityNumber"/>
          </match>
        </readall>
      </derivation>
    </Attribute>

    <!--
    Extrait les mêmes objets de règle que pour "benefitsReadallMatch"
    ci-dessus, mais (selon le stockage des données) peut ne pas être aussi efficace.
    -->
    <Attribute name="benefitsFilterReadall">
      <type>
        <javaclass name="List">
          <ruleclass name="Benefit"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <!-- extraction de tous les objets de règles à partir du

```

A.6.17.54 reference

Permet d'extraire la valeur d'un attribut à partir d'un objet règle.

L'expression `reference` peut contenir une expression enfant qui détermine l'objet règle à partir duquel l'attribut peut être obtenu ; en cas d'omission, l'objet règle contenant l'élément `reference` sera utilisé.

L'expression `reference` constitue un élément clé de la création de règles réutilisables et compréhensibles. Vous pouvez utiliser une référence à un attribut nommé à la place d'une expression. Le valideur du jeu de règles CER renverra une erreur si le type des attributs référencés ne correspond pas au type requis par l'expression.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_reference"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Une référence simple à un autre
      attribut de cette classe de règles -->
    <Attribute name="ageNextYear">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <!-- Cette <référence> ne possède aucun élément enfant,
            ainsi l'objet règle est utilisé pour être "cet objet
règle"-->
          <reference attribute="age"/>
          <Number value="1"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="species">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

A.6.17.55 relatives

Voir le manuel CGIS Household Composition Expressions.

A.6.17.56 removeduplicates

Crée une nouvelle liste en supprimant les éléments en double d'une liste existante.

Si un élément de la liste d'origine apparaît plus d'une fois, seule la première instance est conservée. Sinon, l'ordre des éléments est préservé.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_removeduplicates"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- La liste des relations dans laquelle
         cette personne correspond à "fromPerson". -->
    <Attribute name="relationships">
      <type>
        <javaclass name="List">
          <ruleclass name="Relationship"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Les personnes ayant un lien avec cette personne.

         Les parents n'apparaissent qu'une seule fois dans cette liste,
         même si une personne peut avoir plusieurs liens
         différents avec une autre, par exemple,
         mon grand-père peut également être mon tuteur légal.-->
    <Attribute name="uniqueRelatives">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <removeduplicates>
          <reference attribute="allRelatives"/>
        </removeduplicates>
      </derivation>
    </Attribute>

    <Attribute name="allRelatives">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <!-- obtenez les parents de cette personne en créant une
             liste de "toPerson" à l'autre extrémité de chaque
             relation. -->
        <dynamiclist>
          <list>
            <reference attribute="relationships"/>
          </list>
          <listitemexpression>
            <reference attribute="toPerson">
              <current/>
            </reference>
          </listitemexpression>
        </dynamiclist>
      </derivation>
    </Attribute>

  </Class>

  <!-- Une relation entre deux personnes. -->
  <Class name="Relationship">

    <Attribute name="fromPerson">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>

```

A.6.17.57 ResourceMessage

Crée un message localisable (voir 3.3, «Prise en charge de la localisation», à la page 9) depuis une ressource de propriété.

La propriété peut également spécifier des caractères génériques pour les arguments formatés. La prise en charge ainsi que la syntaxe du formatage sont décrites dans le JavaDoc pour MessageFormat.

avertissement : Comme mentionné dans le JavaDoc, si vous avez besoin de générer un guillemet simple ou une apostrophe ('), vous devez spécifier *deux* guillemets simples dans le texte de propriété ('').

Si vous devez générer un fichier XML ou HTML, et que vous n'avez pas besoin d'un formatage complexe des jetons, ni de modifier le texte du message sans modifier les règles, pensez à utiliser A.6.17.69, «XmlMessage», à la page 239 à la place.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ResourceMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="surname">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="income">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Renvoie un message d'accueil qui peut être
      généré dans l'environnement local de l'utilisateur -->
    <Attribute name="simpleGreetingMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <ResourceMessage key="simpleGreeting"
          resourceBundle="curam.creole.example.Messages"/>
      </derivation>
    </Attribute>

    <!-- Renvoie un message d'accueil contenant
      le titre ainsi que le prénom de la personne.
      Le message d'accueil et le titre sont localisés,
      tandis que le prénom ne l'est pas (il est identique
      dans tous les environnements locaux). -->
    <Attribute name="parameterizedGreetingMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>

```

```
# file curam/creole/example/Messages_en.properties

simpleGreeting=Hello
parameterizedGreeting=Hello, {0} {1}
title.male=Mr.
title.female.single=Miss
title.female.married=Mrs.
incomeStatement=Income: USD{0,number,#0.00}
```

Figure 38. Exemple de propriétés - En anglais

```
# file curam/creole/example/Messages_fr.properties

simpleGreeting=Bonjour
parameterizedGreeting=Bonjour, {0} {1}
title.male=M.
title.female.single=Mlle.
title.female.married=Mme.
incomeStatement=Revenue: EUR{0,number,#0.00}
```

Figure 39. Exemple de propriétés - En français

A.6.17.58 `singleitem`

Permet d'extraire un élément unique d'une liste.

L'expression `singleitem` peut être utile lorsqu'une liste n'est supposée contenir qu'un seul élément, par exemple, lors du filtrage d'une liste selon des critères, qui ne doivent sélectionner qu'un seul élément de la liste.

L'expression `singleitem` permet de définir les éléments suivants :

- **onEmpty**

Le comportement lorsque la liste est vide :

- **error**
Une erreur d'exécution s'est produite (utilisez cette option si la liste ne doit pas être vide) ; ou
- **returnNull**
La valeur *null* est renvoyée.

- **onMultiple**

Le comportement lorsque la liste contient plus d'un élément :

- **error**
Une erreur d'exécution s'est produite (utilisez cette option si la liste ne doit pas contenir plus d'un élément) ;
- **returnNull**
La valeur *null* est renvoyée ;
- **returnFirst**
Le premier élément de la liste est renvoyé ; ou
- **returnLast**
Le dernier élément de la liste est renvoyé.

Pour extraire un élément à partir d'un emplacement spécifique d'une liste, voir `get` dans Annexe B, «Opérations de liste utiles», à la page 247.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_singleitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Le premier né de cette personne -->
    <Attribute name="firstBornChild">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <!-- extrait le premier-né, le cas échéant
          - si aucun enfant, renvoie la valeur null -->
        <singleitem onEmpty="returnNull" onMultiple="returnFirst">
          <!-- trie les enfants par date de naissance -->
          <sort>
            <list alias="child">
              <reference attribute="children"/>
            </list>
            <sortorder>
              <sortitem direction="ascending">
                <reference attribute="dateOfBirth">
                  <current alias="child"/>
                </reference>
              </sortitem>
            </sortorder>
          </sort>

          </singleitem>
        </derivation>
      </Attribute>

    <!-- Extrait l'enregistrement d'informations unique du foyer
      de la mémoire externe - celle-ci doit toujours
      n'en contenir qu'un seul - toute autre valeur entraîne une erreur. -->
    <Attribute name="householdInformation">
      <type>
        <ruleclass name="HouseholdInformation"/>
      </type>
      <derivation>
        <singleitem onEmpty="error" onMultiple="error">
          <readall ruleclass="HouseholdInformation"/>
        </singleitem>
      </derivation>
    </Attribute>

  </Class name="Person">

  <Class name="HouseholdInformation">

```

A.6.17.59 singlepersonhousehold

Voir le guide CGIS Household Composition Expressions.

A.6.17.60 sort

Permet de créer une nouvelle liste en triant les membres d'une liste existante selon un ordre de tri spécifique.

Une expression `sort` permet de définir les éléments suivants :

- **list**
Les éléments existants à trier (qui ne seront pas affectés) ; et
- **sortorder**
L'ordre dans lequel la liste doit être triée.

L'élément `sortorder` permet de définir un ou plusieurs éléments `sortitem`, chacun spécifiant l'élément selon lequel le tri doit être effectué et si le tri doit être effectué dans l'ordre croissant ou décroissant.

Les éléments `sortitem` sont listés par ordre d'importance ; chaque `sortitem` est évalué uniquement si deux éléments triés sont identiques en ce qui concerne l'élément `sortitem` le plus important.

Dans chaque `sortitem`, vous pouvez utiliser A.6.17.13, «current», à la page 184 pour faire référence à l'élément de liste trié. En général, chaque élément `sortitem` fait référence à un attribut ou calcul de l'élément de liste A.6.17.13, «current», à la page 184.

Si deux (ou plus) éléments de la liste sont identiques en ce qui concerne tous les éléments `sortitem`, ils sont alors renvoyés dans le même ordre relatif que la liste source.

Le comportement de l'expression `sort` est similaire à celui de la clause SQL ORDER BY.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sort"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Classe les membres en fonction de leur âge (du plus vieux au plus jeune) ;
      dans le cas des membres ayant le même âge, ceux-ci sont
      classés selon l'ordre alphabétique de leur prénom. -->
    <Attribute name="sortedMembers">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <sort>
          <list>
            <reference attribute="members"/>
          </list>
          <sortorder>
            <sortitem direction="descending">
              <!-- L'âge de la personne dans la liste -->
              <reference attribute="age">
                <current/>
              </reference>
            </sortitem>
            <!-- Le prénom de la personne dans la liste -->
            <sortitem direction="ascending">
              <reference attribute="firstName">
                <current/>
              </reference>
            </sortitem>
          </sortorder>
        </sort>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Integer"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

```

A.6.17.61 specified

Une expression de marqueur permettant d'indiquer que la valeur de l'attribut est spécifiée de façon externe (extraite d'une base de données ou renseignée à l'aide d'un code de test), et non calculée via un traitement des règles.

En général, les attributs `specified` fournissent des informations provenant directement de l'extérieur du système, et d'autres attributs utilisent ces informations externes pour en obtenir de nouvelles.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_specified"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Ces informations ne peuvent être calculées ou obtenues -
         elles doivent être spécifiées depuis une source externe -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Ces informations ne peuvent être calculées ou obtenues -
         elles doivent être spécifiées depuis une source externe -->
    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- D'autres attributs sont susceptibles d'obtenir/calculer davantage
         d'informations en fonction des attributs "specified" ci-dessus -->
  </Class>
</RuleSet>
```

A.6.17.62 String

Une valeur constante de chaîne littérale.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_String"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="StringExampleRuleClass">

    <Attribute name="emptyString">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- Une chaîne vide -->
        <String value=""/>
      </derivation>
    </Attribute>

    <Attribute name="helloWorld">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- La chaîne "Hello, World!" -->
        <String value="Hello, World!"/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>

```

A.6.17.63 sublists

Calcule toutes les sous-listes de la liste fournie, et renvoie ces sous-listes sous forme d'une liste de listes.

Dans le cas d'une liste contenant n éléments, il existe 2^n sous-listes, dont la liste vide et la liste d'origine.

L'ordre des éléments de liste de chacune des sous-listes est identique à celui de la liste d'origine.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sublists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Person"/>
          </listof>
          <members>
            <create ruleclass="Person">
              <String value="Mother"/>
            </create>
            <create ruleclass="Person">
              <String value="Father"/>
            </create>
            <create ruleclass="Person">
              <String value="Child"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Toutes les différentes combinaisons des membres du foyer
-->
    <Attribute name="memberCombinations">
      <!-- Notez que le type est une liste de listes de personnes -->
      <type>
        <javaclass name="List">
          <javaclass name="List">
            <ruleclass name="Person"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <sublists>
          <reference attribute="members"/>
        </sublists>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>

```

Dans cet exemple de jeu de règles, la valeur de *memberCombinations* est calculée comme une liste de ces 8 listes :

- une liste vide (aucun membre du foyer) ;
- Mère ;
- Père ;
- Mère et père ;
- Enfant ;
- Mère et enfant ;
- Père et enfant ;
- Mère, père et enfant (la liste d'origine complète).

A.6.17.64 sum

Calcule la somme numérique d'une liste de valeurs numériques.

Si la liste est vide, cette expression renvoie 0.

La liste des valeurs numériques est généralement fournie par A.6.17.32, «fixedlist», à la page 196 ou A.6.17.16, «dynamiclist», à la page 186.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="netWorth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Exemple d'élément <sum> fonctionnant sur un élément <fixedlist> -->
        <!-- La valeur nette d'une personne correspond à la somme
          de son argent en liquide, de ses économies et de ses actifs -->
        <sum>
          <fixedlist>
            <listof>
              <javaclass name="Number"/>
            </listof>
            <members>
              <reference attribute="totalCash"/>
              <reference attribute="totalSavings"/>
              <reference attribute="totalAssets"/>
            </members>
          </fixedlist>
        </sum>
      </derivation>
    </Attribute>

    <Attribute name="totalAssets">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Exemple d'élément <sum> fonctionnant sur un élément <dynamiclist> -->
        <!-- La valeur totale des actifs d'une personne est obtenue
          via la somme de la valeur de chaque actif -->
        <sum>
          <dynamiclist>
            <list>
              <reference attribute="assets"/>
            </list>
            <listitemexpression>
              <reference attribute="value">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </sum>
      </derivation>
    </Attribute>

    <!-- Les actifs que possède cette personne -->
    <Attribute name="assets">
      <type>
        <javaclass name="List">
          <ruleclass name="Asset"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- NB : cet exemple n'indique pas la manière dont
      le total d'argent en liquide/d'économies est calculé -->
    <Attribute name="totalCash">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>

```

A.6.17.65 this

Une référence à l'objet de règle en cours, analogue au mot-clé `this` dans Java.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_this"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Les animaux domestiques que possède cette personne -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Chaque personne possède deux animaux,
              Skippy et Lassie -->
            <create ruleclass="Pet">
              <!-- définit le propriétaire comme la personne THIS -->
              <this/>
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
              <!-- définit le propriétaire comme la personne THIS -->
              <this/>
              <String value="Lassie"/>
              <String value="Dog"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Initialization>
      <Attribute name="owner">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="species">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>

```

A.6.17.66 Timeline

Crée une chronologie (voir 4.5, «Gestion des données variables au fil du temps», à la page 49) d'un type donné, avec des valeurs valides à partir des dates spécifiées.

Une chronologie doit comporter une valeur à partir du début de la durée (la date null), et pour ce faire, l'expression `Timeline` contient un élément `initialvalue` facultatif permettant de spécifier la valeur du début de la durée. Si elle n'est pas utilisée, alors la collecte d'A.6.17.36, «Intervalle», à la page 198 utilisée *doit* contenir un intervalle comportant une valeur de début null ; dans le cas contraire, une erreur se produit si cette expression est évaluée au moment de l'exécution.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateTimelines">

    <!-- Cet exemple utilise <initialvalue> pour définir la valeur comme valide
      à partir du début de la durée. -->
    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <!-- Valeur à partir du début de la durée -->
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- Les intervalles restants -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2004-12-01"/>
                  </start>
                  <value>
                    <Number value="12000"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>
          </intervals>
        </Timeline>
      </derivation>
    </Attribute>

    <!-- Cet exemple n'utilise pas <initialvalue>. -->
    <Attribute name="aStringTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="String"/>
        </javaclass>

```

A.6.17.67 timelineoperation

Assemble une chronologie (voir 4.5, «Gestion des données variables au fil du temps», à la page 49) à partir d'appels répétés à une expression enfant. En général `timelineoperation` est utilisé avec A.6.17.37, «intervalvalue», à la page 199, et ces deux expressions permettent aux autres expressions d'opérer sur des valeurs des chronologies comme s'il s'agissait de valeurs primitives, puis de rassembler les données obtenues dans une chronologie.

Conseil : Pour chacune des chronologies utilisées comme entrée dans votre algorithme, vous devez généralement encapsuler l'expression renvoyant la chronologie dans un élément A.6.17.37, «intervalvalue», à la page 199, puis encapsuler le résultat total dans un élément `timelineoperation`.

Une brève description de la manière dont `timelineoperation` se comporte au moment de l'évaluation est présentée ci-dessous :

- `timelineoperation` crée un nouveau contexte d'évaluation pour suivre les séries d'appels à effectuer vers son expression enfant (en général, l'expression enfant est appelée plusieurs fois, pour différentes dates) ;
- `timelineoperation` appelle son `expressop`, enfant unique avec une date de contexte null, indiquant le début de la durée ;
- lors de l'évaluation de l'expression enfant (et de ses éléments dépendants), puis dès qu'un élément A.6.17.37, «intervalvalue», à la page 199 est rencontré, puis effectue les actions suivantes :
 - A.6.17.37, «intervalvalue», à la page 199 évalue son expression enfant unique pour obtenir une chronologie, et, à partir de cette chronologie, obtient la valeur de la date correspondant à la date en cours dans le contexte d'évaluation de `timelineoperation` ;
 - A.6.17.37, «intervalvalue», à la page 199 vérifie les autres dates auxquelles la chronologie modifie la valeur, et pour chacune de ces dates, les ajoute à une file d'attente de dates sur laquelle `timelineoperation` doit opérer (lors d'un appel ultérieur) ;
- lorsque le contrôle retourne à `timelineoperation`, une valeur aura été calculée pour une date spécifique, et des dates supplémentaires identifiées auxquelles les chronologies d'entrées modifient la valeur. Pour chaque date, `timelineoperation` appelle à nouveau l'expression enfant (à cette date) jusqu'à ce qu'il n'y ait plus d'autres dates dans la file d'attente.

Le comportement décrit ci-dessus signifie que les expressions internes ne doivent jamais savoir qu'elles font partie du traitement impliquant les chronologies. De plus, le traitement est efficace car les expressions sont uniquement appelées pour les dates auxquelles les chronologies d'entrée modifient la valeur.

Remarque : Si un élément `timelineoperation` opère sur une expression qui n'est *pas* encapsulée par un élément A.6.17.37, «intervalvalue», à la page 199, alors la chronologie obtenue aura toujours une valeur constante.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_timelineoperation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <!--
      true au cours de la vie d'une personne ; false avant la date de naissance,
      et false après la date de décès (le cas échéant)
    -->
    <Attribute name="isAliveTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
      Les actifs que possède la personne, à un moment ou à un autre. Chaque
      valeur d'actif peut varier au fil du temps.
    -->
    <Attribute name="ownedAssets">
      <type>
        <javaclass name="List">
          <ruleclass name="Asset"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
      La valeur totale des actifs d'une personne (ou faisant partie
      de l'héritage d'une personne, si celle-ci est décédée).
    -->
    <Attribute name="totalAssetValueTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <!--
          Totalise la valeur de tous les actifs que possède une personne. La valeur de chaque
          actif peut changer au fil du temps.
        -->

        <!--
          <timelineoperation> crée une chronologie à partir des séries
          de calculs <sum> effectuées dans celle-ci.

          Chaque exécution de <sum> permet de calculer le total d'une
          journée spécifique ; <timelineoperation> assemble ces
          totaux quotidiens dans une chronologie de nombres.
        -->
        <timelineoperation>

          <sum>
            <!--
              Pour chaque actif que possède une personne, obtient sa chronologie de valeurs comptables.
            -->
            <dynamiclist>
              <list>
                <reference attribute="ownedAssets"/>
              </list>
            </sum>
          </derivation>
        </Attribute>
      </RuleSet>

```

Conseil : Si une expression interne renvoie une chronologie, et si vous oubliez d'encapsuler cette expression dans un élément A.6.17.37, «intervalvalue», à la page 199, vous obtiendrez des erreurs de validation CER, telles que présentées dans cet exemple :

```

<!--
  La valeur prise en compte pour les actifs de la personne - autrement dit, la
  valeur de l'actif au cours de la période, lorsqu'il appartient à une personne,
  ou 0 dans le cas contraire.
  -->
<Attribute name="countableValueTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <!--
      rassemble les résultats de chaque appel <choose> dans une
      chronologie
      -->
    <timelineoperation>
      <choose>
        <type>
          <javaclass name="Number"/>
        </type>
        <when>
          <condition>
            <!--
              opère sur chacun des intervalles d'une propriété
              constante
              -->

            <!--
              **** Vous avez oublié d'encapsuler la chronologie renvoyée
              dans <intervalvalue> ****
              -->
            <reference attribute="isOwnedTimeline"/>
          </condition>
          <value>
            <!--
              s'il concerne une date spécifique, l'actif appartient à une personne, puis
              sa valeur comptable à cette date correspond simplement à sa
              valeur
              -->
            <intervalvalue>
              <reference attribute="valueTimeline"/>
            </intervalvalue>
          </value>
        </when>
        <otherwise>
          <value>
            <!--
              s'il concerne une date spécifique, l'actif appartient à une personne, puis
              sa valeur comptable à cette date est égale à zéro
              -->
            <Number value="0"/>
          </value>
        </otherwise>
      </choose>

    </timelineoperation>
  </derivation>
</Attribute>

```

```
ERROR ... Example_timelineoperation.xml(276, 19)
AbstractRuleItem:INVALID_CHILD_RETURN_TYPE: Child 'condition' returns
'curam.creole.value.Timeline<? extends java.lang.Boolean>',
but this item requires a 'java.lang.Boolean'.
```

A.6.17.68 true

La valeur constante booléenne «true».

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_true"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="TrueExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <true/>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <true/>
        </not>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

A.6.17.69 XmlMessage

Crée un message localisable (voir 3.3, «Prise en charge de la localisation», à la page 9) à partir du contenu XML à structure libre.

Le message créé correspond au contenu XML littéral dans l'élément `XmlMessage`, à une exception près : le contenu d'un élément `repl` est défini sur l'expression qu'il contient.

L'élément `repl` comme simple mécanisme de remplacement de jeton ; si vous avez besoin d'un formatage complexe des jetons, ou de modifier le texte de message sans modifier les règles, pensez à utiliser A.6.17.57, «ResourceMessage», à la page 221 à la place.

Remarque : Avant Cúram V6, `XmlMessage` supprimait les espaces entourant les caractères XML intégrés. Dans Cúram V6 et versions ultérieures, `XmlMessage` ne supprime plus les espaces et préserve le format source des caractères XML (en supprimant les commentaires XML).

Si vous avez besoin du comportement de suppression antérieur à Cúram V6 de `XmlMessage`, vous devez alors définir la variable d'environnement d'application `curam.creole.XmlFormat.enableWhitespaceTrimming` sur la valeur `true` dans votre environnement de développement.

Dans un environnement de production, vous devez vous assurer que si la valeur de la variable d'environnement `curam.creole.XmlFormat.enableWhitespaceTrimming` est modifiée de manière dynamique,

vous devez prendre des mesures pour vérifier que les données dérivées dans votre système qui dépendent d'attributs de règles utilisant l'expression XmlMessage doivent être forcées pour recalculer toutes les instances de la valeur d'attribut stockées.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_XmlMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="XmlMessageExampleRuleClass">

    <Attribute name="emptyMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- contains no XML at all -->
        <XmlMessage/>
      </derivation>
    </Attribute>

    <Attribute name="simpleHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- L'utilisation de XmlMessage peut permettre de s'assurer que
          les éléments XML sont démarrés et
          terminés correctement, par exemple, <b> et </b> -->
        <XmlMessage>Le texte suivant s'affiche en gras dans un
          navigateur : <b>Eléments en texte gras.</b>
        </XmlMessage>
      </derivation>
    </Attribute>

    <Attribute name="tokenReplacementHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <XmlMessage><p/>Ce nombre calculé s'affiche en
          italique et est mis en forme en fonction des préférences locales :<i>
            <replace>
              <arithmetic operation="+">
                <Number value="1.23"/>
                <Number value="3.45"/>
              </arithmetic>
            </replace>
          </i>
        <p/>Et voici un message de ressource : <replace>
          <ResourceMessage key="simpleGreeting"
            resourceBundle="curam.creole.example.Messages"/>
        </replace>
        </XmlMessage>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

A.7 Annotations

Depuis Cúram V6, CER prend en charge les "annotations". Une annotation se compose de métadonnées supplémentaires incluses dans un jeu de règles disponible pour les clients de CER, mais n'affecte *pas* le comportement des calculs CER.

Les clients de CER peuvent utiliser des annotations qui régissent leur comportement lors de l'interaction avec les jeux de règles CER.

Les annotations peuvent être placées sur ces éléments dans un jeu de règles CER (bien que chaque annotation puisse contenir une validation pour limiter l'emplacement où il se trouve) :

- jeu de règles (voir A.2, «Jeu de règles», à la page 157) ;
- classe de règles (voir A.4, «Classe de règles», à la page 159) ;
- attribut de règle (voir A.5, «Attribut», à la page 162) ; ou
- expression (voir A.6, «Expressions», à la page 162).

Chaque élément de règle peut contenir une ou plusieurs annotations, voire aucune. Chaque type d'annotation peut apparaître au maximum une fois sur un élément de règle. Par exemple, un jeu de règles peut contenir à la fois une annotation `Label` et une annotation `EditorMetadata`, mais ne peut pas contenir plus d'une annotation `Label`.

A.7.1 Liste alphabétique complète des annotations

Cette section définit toutes les annotations incluses à l'application.

Remarque : Certaines annotations sont destinées à des dérivations spécifiques au métier de l'application. Ces annotations sont toujours incluses ici, mais le lecteur est renvoyé vers d'autres guides Cúram qui décrivent ces annotations dans leur contexte métier.

A.7.1.1 `ActiveInEditSuccessionSetPopulation`

Voir Cúram Advisor Configuration Guide.

A.7.1.2 `Affichage`

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

A.7.1.3 `DisplaySubscreen`

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

A.7.1.4 `EditorMetadata`

Enregistre les informations de diagramme pour un jeu de règles. Géré automatiquement par l'éditeur CER.

Cette annotation peut être spécifiée sur un jeu de règles (voir A.2, «Jeu de règles», à la page 157) .

A.7.1.5 `Indexed`

Obsolète. Inclus à des fins de compatibilité avec les versions antérieures.

`Label`

Fournit une description localisée d'un élément de jeu de règles :

- un jeu de règles (voir A.2, «Jeu de règles», à la page 157) ;
- une classe de règle (voir A.4, «Classe de règles», à la page 159) ;
- un attribut de règle (voir A.5, «Attribut», à la page 162) ; or
- une expression (voir A.6, «Expressions», à la page 162).

Le libellé contient un identificateur (défini par l'éditeur CER) et une description (entrée par l'utilisateur).

Lorsqu'un jeu de règles CER est enregistré ou publié, les valeurs des annotations de libellé dans une règle sont utilisées pour écrire une ressource de propriété (dans l'environnement local de l'utilisateur) dans le magasin de ressources de l'application. A l'inverse, lorsqu'un jeu de règles est affiché dans l'éditeur CER, la ressource de propriétés de l'environnement local de l'utilisateur est extraite du magasin de ressources et utilisée pour renseigner la valeur des annotations de libellé dans le XML du jeu de règles.

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Label"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Description au niveau du jeu de règles -->
    <Label name="Example rule set for labels"
label-id="annotation1"/>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Description au niveau de la classe de règle -->
      <Label name="A Person" label-id="annotation2"/>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Description au niveau de la classe d'attribut -->
        <Label name="Age en cours de la personne, en années"
label-id="annotation3"/>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Description au niveau de l'expression -->
            <Label name="Cette valeur provient directement des informations collectées"
label-id="annotation4"/>
          </Annotations>
        </specified>
      </derivation>
    </Attribute>

    <Attribute name="ageNextBirthday">
      <Annotations>
        <!-- Description au niveau de la classe d'attribut -->
        <Label name="Age de la personne à son prochain
anniversaire, en années"
label-id="annotation5"/>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <Annotations>
            <!-- Description au niveau de l'expression -->
            <Label name="Calcul de l'âge de la personne au prochain anniversaire"
label-id="annotation6"/>
          </Annotations>
          <reference attribute="age">
            <Annotations>
              <!-- Description au niveau de l'expression -->
              <Label name="Obtention de l'âge en cours de la personne"
label-id="annotation7"/>
            </Annotations>
          </reference>
          <Number value="1">
            <Annotations>
              <!-- Description au niveau de l'expression -->
              <Label name="Nombre à ajouter pour obtenir l'âge au prochain
anniversaire" label-id="annotation8"/>
            </Annotations>
          </Number>
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>

```

Legislation

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

SuccessionSetPopulation

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

primary

Identifie l'attribut primaire sur une classe de règle, comme indiqué dans l'éditeur CER.

Cette annotation peut être spécifiée sur une classe de règle (voir A.4, «Classe de règles», à la page 159) uniquement. L'attribut nommé doit être le nom exact d'un attribut déclaré sur la classe de règle (il ne peut pas être utilisé pour nommer un attribut hérité mais pas remplacé).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_primary"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Annotations>
      <!-- Déclaration de l'attribut de règle "primary" pour cette classe de
règle, comme indiqué dans l'éditeur CER -->
      <primary attribute="age"/>
    </Annotations>
    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

relatedActiveInEditSuccessionSet

Voir *Cúram Advisor Configuration Guide*.

relatedEvidence

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

relatedSuccessionSet

Voir le guide *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.

tags

Associe des balises de chaînes arbitraires avec :

- un jeu de règles (voir A.2, «Jeu de règles», à la page 157) ;
- une classe de règle (voir A.4, «Classe de règles», à la page 159) ;

- un attribut de règle (voir A.5, «Attribut», à la page 162) ; ou
- une expression (voir A.6, «Expressions», à la page 162).

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_tags"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamssoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Balises au niveau du jeu de règles -->
    <tags>
      <tag value="Balise au niveau du jeu de règle"/>
      <tag value="Autre balise"/>
    </tags>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Balises au niveau de la classe de règle-->
      <tags>
        <tag value="Balise au niveau de la classe de règle"/>
        <tag value="Autre balise"/>
      </tags>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Balises au niveau de la classe d'attribut -->
        <tags>
          <tag value="Balise d'attribut de règle"/>
          <tag value="Autre balise"/>
        </tags>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Balises au niveau de l'expression -->
            <tags>
              <tag value="Balise d'expression"/>
              <tag value="Autre balise"/>
            </tags>
          </Annotations>
        </specified>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>

```

Annexe B. Opérations de liste utiles

L'expression `property` (voir 7.4.4, «Property», à la page 123 et A.6.17.51, «property», à la page 212) permet d'appeler des méthodes Java "sécurisées".

Les jeux de règles contiennent souvent plusieurs instances de `java.util.List`.

La liste sécurisée des méthodes pour `java.util.List` est incluse au CER :

```
# Safe list for java.util.List

contains.safe=true
containsAll.safe=true

get.safe=true

indexOf.safe=true
isEmpty.safe=true
lastIndexOf.safe=true
size.safe=true
subList.safe=true

# not exposed
hashCode.safe=false
listIterator.safe=false
iterator.safe=false
toArray.safe=false

# mutators - unsafe
add.safe=false
addAll.safe=false
clear.safe=false
remove.safe=false
removeAll.safe=false
retainAll.safe=false
```

Figure 40. Liste sécurisée pour les méthodes `java.util.List`

Tandis que les descriptions de ces méthodes sont disponibles via le JavaDoc pour `java.util.List`, les propriétés les plus utiles sont incluses ici à titre de référence :

- **`isEmpty()`**
Renvoie `true` si cette liste ne contient aucun élément.
- **`size()`**
Renvoie le nombre d'éléments de cette liste.
- **`get(int index)`**
Renvoie l'élément à l'emplacement spécifié de cette liste. Notez que, dans la mesure où CER transmet des valeurs numériques arrondies comme des instances de `Number`, vous devrez utiliser `intValue` pour convertir un élément `Number` en entier.
- **`contains(Object o)`**
Renvoie `true` si cette liste contient l'élément spécifié.

```

<?xml version="1.0" encoding="UTF-8"?>
  <RuleSet name="Example_UsefulListOperations"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
      "http://www.curamsoftware.com/CreoleRulesSchema.xsd">

    <Class name="Person">

      <!-- Une seule personne (dans chaque foyer) sera
      désignée comme le chef de famille -->
      <Attribute name="isHeadOfHousehold">
        <type>
          <javaclass name="Boolean"/>
        </type>
        <derivation>
          <specified/>
        </derivation>
      </Attribute>

      <!-- Les enfants de cette personne. -->
      <Attribute name="children">
        <type>
          <javaclass name="List">
            <ruleclass name="Person"/>
          </javaclass>
        </type>
        <derivation>
          <specified/>
        </derivation>
      </Attribute>

      <!-- Si cette personne a des enfants.

      Tests the isEmpty property of List. -->
      <Attribute name="hasChildren">
        <type>
          <javaclass name="Boolean"/>
        </type>
        <derivation>
          <not>
            <property name="isEmpty">
              <object>
                <reference attribute="children"/>
              </object>
            </property>
          </not>
        </derivation>
      </Attribute>

      <Attribute name="numberOfChildren">
        <type>
          <javaclass name="Number"/>
        </type>
        <derivation>
          <property name="size">
            <object>
              <reference attribute="children"/>
            </object>
          </property>
        </derivation>
      </Attribute>

      <!-- Le deuxième enfant de cette personne, le cas échéant, sinon null -->
      <Attribute name="secondChild">
        <type>
          <ruleclass name="Person"/>
        </type>
        <derivation>
          <!-- Nous devons vérifier si la personne a deux
          enfants ou plus -->
          <choose>
            <type>
              <ruleclass name="Person"/>

```

Annexe C. Utilisation de CER avec le magasin de données

L'application inclut un code d'intégration qui peut créer des objets de règles CER à partir des entrées dans le magasin de données de l'application.

DataStoreRuleObjectCreator est utilisé par le module Universal Access pour convertir les preuves regroupées par un script IEG en objets de règles CER. Cette annexe décrit le fonctionnement de DataStoreRuleObjectCreator.

C.1 DataStoreRuleObjectCreator

DataStoreRuleObjectCreator génère un enregistrement de magasin de données (généralement un enregistrement lié à un utilisateur ou une personne), puis navigue vers tous les enregistrements ultérieurs de cet enregistrement racine (contenant normalement toutes les preuves regroupées relatives à la personne).

Il poursuit par la création d'objets de règles en effectuant un mappage naturel direct entre :

- les types d'entité et les attributs dans le schéma de magasin de données et
- les classes de règles et l'attribut de règle dans le jeu de règles CER.

DataStoreRuleObjectCreator entreprend également une action spéciale pour les attributs de règles CER avec certains noms :

- **parentEntity**

Si une classe de règle contient un attribut de règle appelé `parentEntity`, alors DataStoreRuleObjectCreator définit sa valeur sur l'objet de règle créé à partir de l'enregistrement parent dans le magasin de données (le cas échéant). CER émet une erreur d'exécution si le type de cet attribut de règle ne correspond pas à la classe de règle de l'objet de règle de l'entité parent.

- **childEntities_<nom de classe de règle>**

Si une classe de règle contient des attributs appelés `childEntities_` suivis du nom d'une classe de règle, DataStoreRuleObjectCreator définit la valeur de chacun de ces attributs sur une liste d'objets de règles créés à partir des enregistrements enfant de ce type dans le magasin de données. CER émet une erreur d'exécution si le type de cet attribut de règle n'est pas une liste de la classe de règle nommée.

C.2 Exemple

Pour mieux comprendre le comportement de DataStoreRuleObjectCreator, prenons un exemple. Cet exemple se base sur le module Universal Access.

Un script IEG peut capturer les données sur le revenu d'un ménage. Le ménage peut contenir un nombre de personnes, chaque personne pouvant avoir plusieurs informations relatives à ses revenus.

Voici une vue simplifiée de la structure du schéma Datastore :

- Application:
 - Person (0..n):
 - firstName (String)
 - lastName (String)
 - Income (0..n):
 - type (Code from the IncomeType code table)
 - amount (Number)

John effectue son auto-sélection et enregistre les informations relatives à son ménage (John et son épouse Mary) et ses revenus (John est sans-emploi, Mary a deux emplois à temps partiels). Les informations de John sont stockées dans Datastore :

- Application #1234:
 - Person #1235:
 - firstName: John
 - lastName: Smith
 - Income: <no records>
 - Person #1236:
 - firstName: Mary
 - lastName: Smith
 - Income #1237:
 - type: Part-time
 - amount: 25
 - Income #1238:
 - type: Part-time
 - amount: 30

Le jeu de règles CER configuré pour être utilisé avec ce type de sélection contient des classes de règles comme suit (NB/// aucune classe de règles n'est affichée) :

```

<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="DataStoreMappingExample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- NB/// aucune classe de règles pour le type d'entité CDS "Application" ;
    les attributs stockés directement sur une application ne sont
    pas requis par les règles. La création d'un objet de règle
    qui ne sera pas utilisé ne pose donc pas de problème. -->

  <!-- Le nom de cette classe de règles correspond à celui du type
    d'entité CDS -->
  <Class name="Person">
    <!-- Le nom de cet attribut de règle correspond à celui d'un
      attribut sur le type d'entité CDS. Sa valeur sera par conséquent
      spécifiée automatiquement par
      DataStoreRuleObjectRetriever. -->
    <Attribute name="firstName">
      <!-- Le type d'attribut de règle doit se conformer au
        type d'attribut CDS. Sinon, CER indique
        une erreur d'exécution. -->
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- NB/// aucun attribut de règle pour l'attribut CDS de lastName.
    -->

    <!-- Les attributs de règle correspondant au modèle <nom de la classe
    de règles childEntities_>"
      seront traités spécialement par
      DataStoreRuleObjectRetriever.

      DataStoreRuleObjectRetriever spécifie la valeur de
      cet attribut comme valeur de tous les objets de règle créés à partir
      des enregistrements Income enfant qui appartiennent à l'enregistrement de cette personne
      dans le fichier CDS. -->
    <Attribute name="childEntities_Income">
      <!-- Le type doit être une liste d'objets de règle Income -->
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Le nom de cette classe de règles ne correspond pas à un type d'entité
  CDS.
  Par conséquent, DataStoreRuleObjectRetriever ne crée pas d'objets de règle
  pour cette classe de règles. -->
  <Class name="Benefit">
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <Class name="Income">
    <!-- Un attribut de règle nommé "parentEntity" sera
    traité spécialement par

```

Lorsque John effectue son auto-sélection, le module Universal Access charge le jeu de règles ci-dessus et crée une session CER.

Le module Universal Access appelle `DataStoreRuleObjectCreator` et indique l'enregistrement Datastore (Application #1234).

`DataStoreRuleObjectCreator` extrait tous les enregistrements descendants de l'application #1234 et les traite comme suit :

- **Application #1234**
Ignoré, comme il n'y a pas de classe de règles nommée "Application" dans le jeu de règles ;
- **Person #1235**
crée une instance d'objet de règle de la classe de règles Person :
 - **firstName**
Défini sur "John" ;
 - **Nom de famille**
Ignoré, comme il n'y a pas d'attribut de règle nommé "lastName" dans la classe de règles Person ;
 - **childEntities_Income**
Défini comme liste vide, car il n'existe pas d'enregistrements Income pour l'enregistrement Person #1235 ;
- **Person #1236**
crée une instance d'objet de règle de la classe de règles Person, avec :
 - **firstName**
Défini sur "Mary" ;
 - **Nom de famille**
Ignoré ;
 - **childEntities_Income**
Défini comme liste contenant les deux objets de règle Income de Mary (créés ci-dessous) ;
- **Income #1237**
crée une instance d'objet de règle de la classe de règles Income, avec :
 - **parentEntity**
Défini comme objet de règle Person créé (ci-dessus) pour Mary à partir de l'enregistrement Person #1236 ;
 - **type**
Défini comme code "Part-time" ;
 - **amount**
Défini comme nombre "25" ;
 - **(employerName)**
Non spécifié, car l'entité Datastore ne possède pas d'attribut nommé "employerName" ;
- **Income #1238**
crée une instance d'objet de règle de la classe de règles Income, avec :
 - **parentEntity**
Défini comme objet de règle Person créé (ci-dessus) pour Mary à partir de l'enregistrement Person #1236 ;
 - **type**
Défini comme code "Part-time" ; et
 - **amount**
Défini comme nombre "30".

Enfin, le module Universal Access demande au jeu de règles ses questions standard sur les programmes, l'éligibilité et l'explication ; les classes de règles des programmes (non affichés dans l'exemple ci-dessus) permettent d'accéder aux objets de règle (créés par `DataStoreRuleObjectCreator`) lors de la réponse à ces questions.

Pour plus d'informations sur le Datastore d'application et ses schémas, voir le guide `Creating Datastore Schemas`.

Annexe D. Conformité

Cette annexe explique comment procéder au développement dans le respect de la conformité. En suivant ces indications, les clients pourront plus facilement effectuer la mise à niveau vers les versions ultérieures de l'application.

D.1 API publique de CER

L'infrastructure CER possède une interface de programme d'application (API) publique que vous pouvez appeler dans vos tests de jeu de règles et code d'application. Les exemples de ce manuel expliquent la méthode d'utilisation de plusieurs fonctions de l'API publique de l'infrastructure CER. L'application ne modifie ni ne supprime aucun élément dans cette API publique si les normes en matière de gestion de l'impact client ne sont pas respectées.

Sauf en cas d'autorisation explicite dans JavaDoc, vous ne devez *pas* fournir votre propre implémentation d'une interface Java CER ni sous-classer une classe Java d'implémentation CER.

D.1.1 Identification de l'API

L'outil JavaDoc fourni avec CER constitue le seul moyen d'identifier les classes, interfaces et méthodes publiques qui composent l'API publique de CER.

D.1.2 Hors de l'API

L'infrastructure CER contient également certaines classes, interfaces et méthodes publiques, qui ne font *pas* partie de l'API.

Important : Pour être conforme, vous ne devez *pas* créer de dépendance à une classe ou une interface CER, ni appeler de méthode, autres que celles décrites dans JavaDoc.

Les classes, interfaces et méthodes CER hors de l'API publique peuvent être modifiées ou supprimées sans préavis.

Sauf en cas d'autorisation explicite dans JavaDoc, vous ne devez *pas* placer vos propres classes ou interfaces dans le module `curam.creole` ou l'un de ses sous-modules.

D.2 Expressions CER

Seules les expressions CER répertoriées dans ce document sont prises en charge. Voir A.6.17, «Liste alphabétique complète des expressions», à la page 164 pour connaître les expressions prises en charge.

Remarque : Le schéma du jeu de règles fourni avec CER contient également des expressions non prises en charge. Ces expressions sont fournies à titre d'information et peuvent être modifiées ou supprimées sans préavis. N'utilisez pas d'expressions CER non prises en charge dans vos jeux de règles.

D.3 Jeux de règles compris avec l'application

Certains composants de l'application peuvent inclure des jeux de règles CER susceptibles de posséder leurs propres exigences de conformité. Ces exigences de conformité concernant les jeux de règles CER introduites par les composants de l'application n'entrent pas dans le cadre du présent document.

Consultez la documentation accompagnant ces composants pour savoir si vous êtes autorisé à personnaliser le jeu de règles CER, et le cas échéant, connaître les restrictions qui s'appliquent en matière de personnalisation.

Le jeu de règles RootRuleSet fourni avec CER ne doit pas être modifié par les clients.

D.4 Exemples contenus dans ce guide

Les exemples d'artefacts contenus dans ce guide (jeux de règles, code Java et fichiers de propriétés) peuvent être modifiés ou supprimés sans préavis.

Vous êtes libre de *copier* ces exemples d'artefacts pour votre usage personnel ; cependant, il convient de noter qu'aucune prise en charge de mise à niveau n'est fournie pour ces exemples d'artefacts.

D.5 Tables de base de données de CER

L'infrastructure CER inclut un certain nombre de tables de base de données. En général, ces tables sont internes à CER et les données qu'elles contiennent peuvent être lues ou écrites uniquement via l'interface de programme d'application (API) publique de CER.

Pour plus de détails sur les restrictions de lecture/écriture relatives à ces tables de base de données, voir les sous-sections suivantes.

Remarque : Toutes les tables de base de données de l'infrastructure CER sont préfixées par le mot CREOLE ; toutefois, l'inverse n'est pas vrai.

Il existe des tables préfixées par CREOLE qui font partie d'autres composants d'application et sont soumises à leurs propres instructions de conformité. Seules les instructions de conformité concernant les tables de base de données de l'*infrastructure CER* sont décrites ci-après.

D.5.1 CREOLERuleSet

Cette table de base de données stocke une ligne pour chaque jeu de règles CER publié dans le système.

Généralement, l'accès en lecture et en écriture à cette table de base de données est restreint à l'API publique de CER uniquement ; cependant, vous êtes autorisé à utiliser le gestionnaire de données de l'application pour remplir cette table de base de données, à condition que les critères suivants soient respectés :

- La valeur de la colonne name doit correspondre au nom du jeu de règles tel que défini dans le langage XML de la colonne ruleSetDefinition ; et
- La valeur de la colonne ruleSetVersion doit être null.

Figure 41. Exemple d'entrée provenant d'un fichier CREOLERuleSet.dmx conforme

Voir le guide Working with Cúram Express Rules pour connaître les étapes d'extraction des données CREOLERuleSet à partir de votre application (et des données AppResource associées, si nécessaire).

D.5.2 CREOLEMigrationControl

CREOLEMigrationControl est une table de contrôle à une seule ligne qui est utilisée pour empêcher la publication simultanée de jeux de règles CER.

Les données sur cette table sont internes à CER et ne peuvent être lues ou écrites par aucun autre composant.

La ligne unique sur cette table est renseignée via un fichier DMX fourni avec l'application. Les clients ne doivent pas modifier ce fichier DMX ni créer d'autres fichiers DMX qui ciblent la table CREOLEMigrationControl.

D.5.3 Autres tables de base de données de l'infrastructure CER

Les autres tables de base de données incluses avec l'infrastructure CER sont les suivantes :

- CREOLEAttributeAvailability ;
- CREOLEAttributeInheritance ;
- CREOLERuleAttribute ;
- CREOLERuleAttributeValue ;
- CREOLERuleClass ;
- CREOLERuleClassInheritance ;
- CREOLERuleObject ;
- CREOLERuleSetDependency ;
- CREOLERuleSetEditAction ;
- CREOLERuleSetSnapshot ; et
- CREOLEValueOverflow.

Ces tables de base de données de l'infrastructure CER sont soumises à la condition de conformité générale suivante : les données sur ces tables de base de données ne doivent pas être lues ou écrites autrement que via l'API publique de CER. Par ailleurs, le remplissage initial de ces tables de base de données via les fichiers DMX n'est *pas* pris en charge.

D.6 Gestionnaire de dépendance

Le gestionnaire de dépendance est interne à Cúram ; l'accès depuis un code personnalisé et la personnalisation ne sont pas pris en charge.

Tous les artefacts appartenant au gestionnaire de dépendance sont contenus dans le module de code `curam.dependency` et ses sous-modules¹⁹. Vous ne devez pas placer vos propres classes ou interfaces dans le module de code `curam.dependency` ou l'un de ses sous-modules.

Les tables de base de données suivantes appartiennent au gestionnaire de dépendance :

- Dependency ;
- PrecedentChangeSet ;
- PrecedentChangeItem ; et
- PrecedentChangeSetBatchCtrl.

Ces tables de base de données ne doivent pas être personnalisées n'importe comment. Par ailleurs, les données sur ces tables de base de données ne doivent pas être lues ou écrites autrement que par le gestionnaire de dépendance lui-même.

Le remplissage initial de ces tables de base de données via les fichiers DMX n'est *pas* pris en charge, à l'exception des fichiers DMX fournis avec l'application. De plus, il n'est pas possible de personnaliser ou d'omettre le remplissage de ces tables de base de données via les fichiers DMX compris avec l'application.

19. Certains artefacts dans ce module de code bénéficient de la contribution de CER et d'autres de l'application de base.

Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM. IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A. Pour le Canada, veuillez adresser votre courrier à : IBM Director of Commercial Relations IBM Canada Ltd 3600 Steeles Avenue East Markham, Ontario L3R 9Z7 Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

Licence sur la propriété intellectuelle

Mentions légales et droit de propriété intellectuelle.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japon

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. INTERNATIONAL BUSINESS MACHINES CORPORATION FOURNIT CETTE PUBLICATION "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, EXPLICITE OU IMPLICITE, Y COMPRIS NOTAMMENT, LES GARANTIES IMPLICITES DE NON-CONTREFAÇON, DE QUALITE MARCHANDE OU D'ADEQUATION A UN USAGE PARTICULIER. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies. Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A. Pour le Canada, veuillez adresser votre courrier à : IBM Director of Commercial Relations IBM Canada Ltd 3600 Steeles Avenue East Markham, Ontario L3R 9Z7 Canada

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM, conformément aux dispositions du Livret contractuel, des Conditions Internationales d'Utilisation de Logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles.

IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Tous les tarifs indiqués sont les prix de vente actuels suggérés par IBM et sont susceptibles d'être modifiés sans préavis. Les tarifs appliqués peuvent varier selon les revendeurs.

Ces informations sont fournies uniquement à titre de planification. Elles sont susceptibles d'être modifiées avant la mise à disposition des produits décrits.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

LICENCE DE COPYRIGHT :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de

copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes. Les exemples de programmes sont fournis "EN L'ETAT", sans garantie d'aucune sorte. IBM décline toute responsabilité relative aux dommages éventuels résultant de l'utilisation de ces exemples de programmes.

Toute copie intégrale ou partielle de ces exemples de programmes et des oeuvres qui en sont dérivées doit inclure une mention de droits d'auteur libellée comme suit :

© (nom de votre société) (année). Des segments de code sont dérivés des exemples de programmes d'IBM Corp.

© Copyright IBM Corp. _entrez l'année ou les années_. All rights reserved.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Marques

IBM, le logo IBM et [ibm.com](http://www.ibm.com) sont des marques ou des marques déposées d'International Business Machines Corp. dans de nombreux pays. Les autres noms de produits et de services peuvent être des marques d'IBM ou d'autres sociétés. Une liste des marques commerciales actuelles d'IBM est disponible sur Internet sous "Droits d'auteur et marques" à l'adresse <http://www.ibm.com/legal/us/en/copytrade.shtml>.

D'autres noms peuvent être des marques de leurs propriétaires respectifs. Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.



Imprimé en France