Gestion des règles « métier »

VINCENT LEGENDRE, GÉRALD PETITJEAN ET THIERRY LEPATRE

Résumé : L'objet de cet article est de présenter les principes des BRMS (Systèmes de Gestion de Règles « Métier ») : leur évolution depuis les systèmes experts des années 1980, leurs objectifs, leurs composants et leur architecture. Quant au fonctionnement des BRMS, il sera décrit au travers de la définition d'un langage « métier », de l'écriture des règles, et de l'exécution du moteur d'inférence. Les principaux outils de BRMS et leurs applications les plus courantes seront rapidement présentés. Nous évoquerons également les moteurs de règles spécifiques, développés directement dans des langages tels que C++, destinés à être appelés dans des algorithmes d'optimisation et donc soumis à de fortes contraintes de temps de réponse. Enfin, nous aborderons, au travers d'un exemple, le développement de moteurs de règles reposant sur Prolog.

Mos clés : Règles « métier », BRMS, SOA, systèmes experts, Prolog, optimisation

1. PRÉAMBULE

1.1 HISTORIQUE : LES SYSTÈMES EXPERTS

Connus sous l'acronyme anglophone BRMS (« Business Rules Management Systems »), les outils de gestion de règles métier sont les descendants des systèmes experts, systèmes qui ont connu leur heure de gloire dans les années 1980. Il s'agissait d'outils capables de reproduire les raisonnements, les comportements et les mécanismes cognitifs d'un expert humain dans un domaine particulier. Les systèmes experts sont donc des outils utilisant des techniques d'intelligence artificielle afin de fournir une aide à la décision.

Un système expert est constitué de trois parties :

- une base de faits,
- une base de règles,
- et un moteur d'inférence, qui utilise les faits et les règles afin de déduire de nouveaux faits jusqu'à donner une réponse à une question posée initialement.

L'ensemble constitué de la base de faits et de la base de règles est aussi appelé la base de connaissances.

Le moteur d'inférence a pour fonction d'exploiter la base de connaissances afin de résoudre le problème qui lui est posé. Il existe deux types d'algorithmes :

- les algorithmes à chaînage avant : les règles sont appliquées sur une base de faits initiale, et vont modifier cette base (ajouts, suppressions, modifications). Ces modifications impliquent le déclenchement d'autres règles (ce qui constitue le chaînage avant) et ainsi de suite jusqu'à épuisement des déductions possibles. Ce type d'algorithme a été utilisé, par exemple, dans des systèmes de diagnostics médicaux pour déduire des pathologies probables à partir de symptômes constatés.
- les algorithmes à chaînage arrière : le moteur cherche dans ce cas à vérifier une règle (appelée un « but ») en fonction de sa base de connaissances, soit directement via un fait existant, soit via d'autres règles (ce qui constitue le chaînage arrière). Ce type d'algorithme est utilisé notamment pour le traitement du langage naturel.

Toute la difficulté de la mise en place d'un système expert réside dans l'acquisition des connaissances d'un expert et de leur formalisation en tant que faits et règles.

Finalement, ces systèmes n'ont pas été développés massivement, leur mise au point technique (codage et intégration) étant difficile et leur maintenance étant particulièrement lourde. Les outils de BRMS ont précisément pour objectifs de faciliter à la fois le développement et l'intégration de tels systèmes, ainsi que leur maintenance et leur évolutivité.

1.2 PRINCIPES DES BRMS

1.2.1 Définitions : règle « métier » et BRMS

Une règle « métier » est une description de haut niveau, permettant de contrôler et/ou de prendre

une décision, en utilisant des concepts propres à une entreprise ou une organisation. Ainsi, les règles « métier » décrivent ce qu'un expert doit faire pour prendre une décision, c.-à-d. la connaissance « métier » des experts. La modélisation interne doit donc permettre :

- d'exprimer simplement les concepts métiers (objets),
- de décomposer en règles simples des éléments de raisonnement,
- d'organiser (ordonner, affecter des priorités, brancher) l'exécution des règles ou de paquets de règles.

L'ensemble des règles formalise alors la connaissance des experts, et permet d'automatiser un raisonnement global complexe impossible à implémenter avec des langages impératifs classiques (C++, Java, .Net, PL/SQL, …).

1.2.2 Objectifs

La formalisation sous forme de règles permet de capitaliser et d'automatiser le savoir-faire des experts (politique commerciale, marketing, gestion des risques, …), tout en apportant une standardisation et une auditabilité de ces règles « métier », facilitant alors leur consultation et leur maintenance par le plus grand nombre. Toute implémentation, même via une interface la plus ergonomique qui soit, implique forcément des contraintes syntaxiques, d'organisation et de logique qui vont formater et organiser la connaissance. Cela peut également amener assez souvent des remises en cause, par exemple si deux experts se rendent compte que sur un point précis, leurs règles sont contradictoires.

La sémantique d'une règle standard (« Si …. Alors … ») est particulièrement bien adaptée à ce que l'on peut trouver dans les réglementations ou dans les documents de spécifications, ce qui permet d'avoir des délais de mise en œuvre et de validation plus courts que pour un projet écrit dans un langage classique, le passage de la spécification aux règles étant naturel. Les règles sont donc beaucoup plus lisibles que du code C++ ou JAVA. Le raisonnement global complexe est ainsi décomposé en règles simples et lisibles (au moins autant qu'un document de spécifications), et non noyées dans un code source complexe compréhensible uniquement par des informaticiens (voire de l'informaticien qui l'a écrit).

L'objectif principal d'un BRMS est de fournir un environnement d'édition des règles utilisable simplement et directement par des experts « métier » sans compétences informatiques particulières. Cela permet de faciliter la maintenance évolutive, grâce à la lisibilité des règles, et induit donc une meilleure réactivité et adaptation aux changements. En effet, le codage et la validation des nouvelles règles sont plus rapides à effecteur et plus aisées à mettre en œuvre, les utilisateurs n'ayant plus besoin de faire appel aux services informatiques. Ceci est possible, car les BRMS sont conçus pour séparer la logique « métier » (la formalisation des règles, en utilisant directement les concepts « métier ») de la logique système (la mise en œuvre technique).

1.2.3 Composants / Architecture

Un BRMS est un agrégat de composants interagissant entre eux :

- Le moteur d'inférence :
  - API (Java, .Net, C++, …) permettant de manipuler les faits et les règles, et d'exécuter une inférence
  - Les règles sont écrites dans un langage propre au produit, reprenant en général les principes historiques des premiers systèmes en LISP
  - L'exécution se base généralement sur l'algorithme RETE et ses améliorations pour les systèmes en chaînage avant.
- Un environnement de développement (IDE) :
  - Permet d'éditer les règles, fonctions et objets du système
  - Définition de la surcouche métier, et plus généralement de toute l'infrastructure utilisée par les experts métier pour écrire les règles
  - Ilog JRules, JBoss Rules – Drools : utilisation d'Eclipse (avec un plug-in particulier)
  - BlazeAdvisor : IDE dédié

- Une interface Web d'édition de règles orienté utilisateurs « métier » :
  - Permet d'éditer des règles (partagées sur un serveur) via une interface web
  - Permet la gestion des versions des règles et le déploiement.

Les deux premiers composants sont réservés à un usage par des experts informatiques confirmés. La troisième partie, c'est-à-dire l'interface Web, constitue véritablement la « vitrine » d'un BRMS et est destinée à être utilisée par les experts « métier », Sur ce point, les outils commerciaux sont plus fournis que les outils gratuits, les éditeurs concentrant leurs efforts sur l'ergonomie et les fonctionnalités proposées :

- Gestion des versions des règles, traçabilité de leurs modifications,
- Gestion du cycle de vie (développement, validation, déploiement en production),
- Recherche de règles (selon des critères avancés)
- Documentation (rapports, audits …)
- Tests, contrôles de cohérence
- Droits d'accès des utilisateurs
- …

L'architecture classique d'une application utilisant un BRMS est représentée dans le schéma suivant :

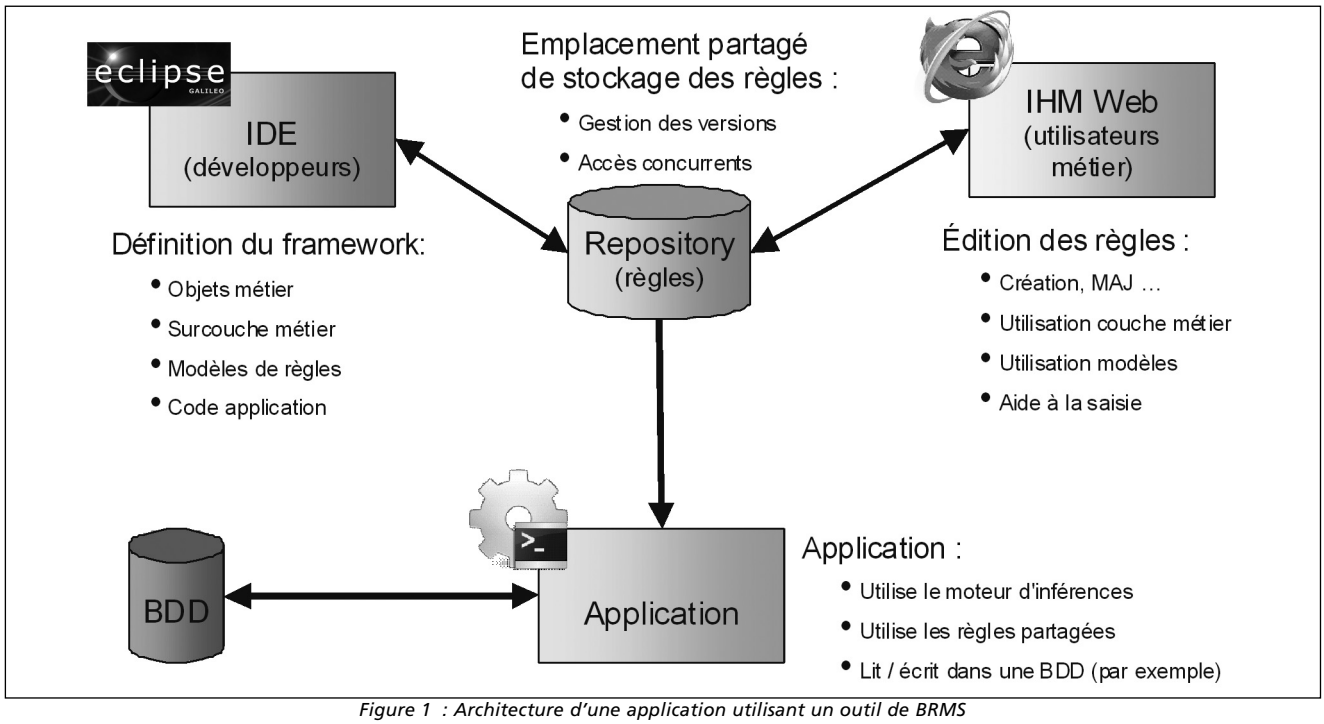

Figure 1 : Architecture d'une application utilisant un outil de BRMS

1.3 Les étapes d'un projet BRMS

Un projet de BRMS se déroule selon les étapes suivantes :

1. Phase de conception / paramétrage du BRMS
   - Définition des concepts métier / langage utilisé pour les règles
   - Définition des objets physiques / lien avec le modèle métier
   - Définition de modèles de règles
2. Écriture des règles
   - Par l'équipe de développement dans un premier temps puis par l'utilisateur « métier »
   - Tests et validation
3. Phase de maturité : modification, ajout de règles
   - S'appuyant sur des données existantes dans le SI : l'utilisateur « métier » est autonome
   - Nécessitant de nouvelles données : l'intervention de l'équipe de développement est alors nécessaire
4. Intégration
   - Utilisation native du langage Java
   - Intégration dans des architectures de type EJB et SOA.

Toutefois, certains pré-requis et contraintes doivent être respectés :

- La couche d'objets « métier » doit être suffisamment stable avant d'écrire les règles.
- Il est préférable d'avoir recours à une courte période de consulting de la part d'un expert en BRMS, ce qui permet d'accélérer la mise en place de l'architecture.
- Les première phases d'écriture de règles sont à réaliser par un développeur, puis le transfert doit être progressif vers l'utilisateur « métier ».

Le point critique d'un projet utilisant les BRMS concerne le développement de tout ce qui va être exposé aux utilisateurs « métier », c'est-à-dire le langage « métier » utilisé, mais aussi l'élaboration de modèles et d'autres fonctions connexes comme des tests de non-régression ou des aides au débogage. La criticité vient principalement du niveau technique des utilisateurs « métier » concernés :

- Si trop de liberté est laissée et que les utilisateurs ne sont pas assez formés, le système peut devenir instable, incompréhensible et donc ne pas remplir du tout ses objectifs d'amélioration de d'évolutivité du système, les utilisateurs ayant systématiquement besoin de l'assistance de l'équipe informatique.
- À l'inverse, si le système est trop contraint, les utilisateurs ne pourront pas forcément exprimer toutes les règles qu'ils souhaiteraient. Dans ce cas, soit les utilisateurs cessent d'utiliser le système, soit les demandes à l'équipe informatique se multiplient.

1.4 Les principaux outils de BRMS

Nous citerons ici les principaux outils de BRMS que nous avons rencontrés lors de projets industriels. Ceci ne constitue absolument pas une liste exhaustive de tous les outils existants basés sur des règles. Notamment, les trois produits cités sont basés sur le langage Java, mais il en existe d'autres utilisant des langages comme le C++ ou le Lisp.

Les outils les plus complets sont bien évidemment les outils commerciaux : *JRules* (anciennement ILOG, et maintenant IBM Websphere) et *Blaze Advisor* (FICO, Fair Isaac Corporation). Ces deux outils utilisent des dérivés de RETE, améliorés au fil des années, et fournissent donc de très bonnes performances. De plus, les outils mis à la disposition des développeurs et des utili-

sateurs métiers sont très complets, et éprouvés depuis des années sur de nombreux projets industriels.

Une solution Open-Source existe également et commence à être utilisée pour des applications industrielles : *Drools* (ou *JBoss Rules*, développée par Red Hat). Cet outil, de par sa gratuité, offre beaucoup moins de fonctionnalités que les outils commerciaux, et est moins performant sur de gros volumes (données et/ou règles). Cependant, les bases sont là, y compris au niveau des interfaces « métier », et l'évolution est rapide et prometteuse.

2. Fonctionnement des BRMS

2.1 Définition d'un langage « métier »

L'objectif principal d'un BRMS est de permettre à des experts « métier », non informaticiens, de modifier les règles présentes dans le système de manière autonome. Il faut donc impérativement que l'édition des règles utilise les concepts « métier » habituellement manipulés par ces experts, en « masquant » leur implémentation informatique. Pour ce faire, les BRMS permettent de définir un langage « métier », manipulé directement par les éditeurs de règles, avec pour chaque concept « métier » sa correspondance en termes de code informatique. Sauf exception, ce langage est en fait une mise en correspondance entre le modèle conceptuel de données (MCD) et le modèle physique de données (MPD).

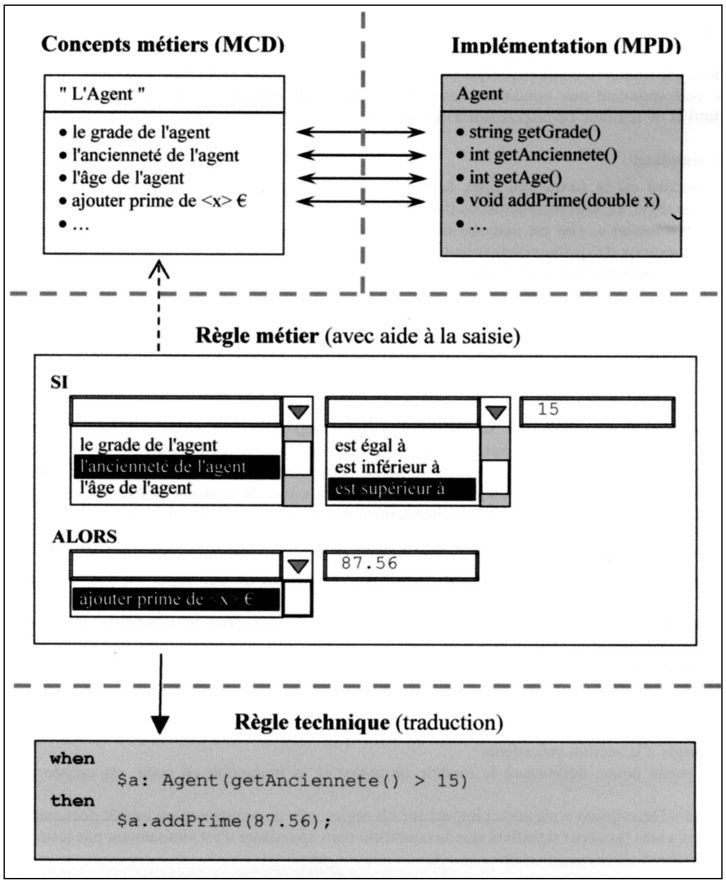

Figure 2 : Définition du langage « métier » et des règles

2.2 Écriture des règles

Une règle est constituée de :

- Une partie de conditions (« Si l'agent A a un grade G »)
- Une partie actions, exécutée si les conditions sont vérifiées (« Alors attribuer à A la prime P d'un montant de X € »)

Les conditions portent sur les objets présents en mémoire de travail du moteur et leurs attributs. Les actions sont une série de commandes sur les objets récupérés dans les conditions. Une même règle sera exécutée pour l'ensemble des objets vérifiant ses conditions.

La plupart des outils proposent trois façons de définir des règles :

- Règles standard
  - Si … Alors … classique, utilisant ou non le langage « métier », avec plus ou moins d'aide à la saisie contextuelle selon les outils.
- Tables de décision
  - Édition synthétique de plusieurs règles partageant le même modèle
  - Colonnes de conditions, puis colonnes d'actions
  - 1 ligne = 1 règle
- Arbres de décision
  - Un arbre de décision est analogue aux tables, mais organisé en arbre. Il permet de « contextualiser » des conditions par rapport à d'autres (ce qui donnerait beaucoup de cases vides dans une table). Il est composé de nœuds, correspondant aux conditions (attributs à tester), de branches (une par valeur possible pour une condition) et de feuilles, correspondant à des actions.

2.2.1 Règle standard

Une règle standard est la forme qui offre le plus de possibilités. La syntaxe est proche du fonctionnement historique d'une règle, et, sauf mise à disposition d'une surcouche « métier » bien étudiée et une bonne formation des utilisateurs « métier », elle est inaccessible à des non-développeurs. En effet, même si c'est la forme qui permettrait aux experts d'exprimer pleinement leur créativité, cette souplesse entraîne en général des problèmes, car elle peut assez facilement introduire des erreurs et rendre le système inutilisable.

Une première alternative consiste à créer des modèles de règles, en figeant certaines parties d'une règle standard. Ces modèles de règles servent alors de base aux utilisateurs métiers pour créer de nouvelles règles, en modifiant uniquement ce qui a été prévu et en limitant ainsi le risque d'écrire une règle erronée.

2.2.2 Tables de décision

Assez souvent, les règles « métier » se ressemblent beaucoup dans leur structure (les tests et les

	Primes Ancienneté			
Type colonne	CONDITION	CONDITION	CONDITION	ACTION
Objet	a:Agent			
Test / Action	getAnciennete() >= $param	getAnciennete() < $param	getAge() >= $param	a.addPrime($param);
Description	**Ancienneté minimum**	**Ancienneté maximum**	**Age minimum**	**Prime**
Jeunes		5		0
Normaux	5	10		50
Anciens séniors	10	20	50	120
Anciens				100
Très anciens	20			200

Figure 3 : Exemple de table de décision

actions effectués), avec quelques valeurs différentes. L'utilisation de modèles est alors fastidieuse dans le sens où chaque jeu de valeurs requiert l'écriture d'une nouvelle règle. Dans ces cas, on utilise les tables de décision, qui sont un moyen de créer plusieurs règles basées sur le même modèle mais avec des valeurs différentes, le tout regroupé de manière synthétique dans un tableau (forme généralement proche des documents de spécifications).

Les principes sont les suivants :

- Une table est basée sur un (unique) modèle de règle, autrement dit une règle « à trous » représentant ses paramètres, dont les valeurs changent selon les cas de figure.
- Les colonnes représentent des paramètres. On trouve des colonnes pour les conditions, et d'autres pour les actions, ayant chacune leur « traduction » en code.
- Chaque ligne d'une table va générer une nouvelle règle, en utilisant les valeurs des cellules de ligne. Par convention, les éléments du modèle correspondant aux cellules vides sont ignorés

Voici un exemple illustrant le principe des tables (fortement inspiré de Drools), basé sur le modèle objet métier donné en exemple à la section précédente :

- Toute la partie haute, définissant le modèle de règles et sa traduction en code, est cachée à l'utilisateur « métier »
- La colonne « Description » n'a aucun impact sur les règles, elle a uniquement un intérêt documentaire
- Les cellules vides (grisées) signifient que la condition correspondante n'est simplement pas testée

On peut apprécier ici la simplicité de manipulation d'une telle structure, ainsi que le contrôle sur les règles générées (le modèle cadre les possibilités et empêche l'apparition de bugs graves). La simplicité d'édition est encore plus flagrante pour des outils commerciaux disposant de toute une panoplie d'aides à la saisie, ou de cases spéciales « sinon », signifiant « toutes les conditions non testées avant » bien utiles pour garantir la complétude des règles générées.

Cette simplicité a une contrepartie, à savoir qu'une table est relativement figée. En effet, il est très facile d'ajouter une règle avec de nouvelles valeurs, mais il est impossible pour un utilisateur « métier » d'ajouter une nouvelle condition ou une nouvelle action sans intervention de l'équipe informatique. Cela étant, cette forme de modélisation est largement utilisée dans les projets de BRMS, en palliant l'aspect figé par la fourniture d'un large éventail de modèles de tables différentes.

2.2.3 Arbres de décision

Il est également possible de modéliser des règles sous forme d'arbre (Blaze Advisor a également introduit récemment la notion de graphe). Dans ce type de modélisation, les nœuds sont des tests, les branches sont les valeurs à tester et les feuilles les actions. Cela permet d'effectuer certains tests uniquement sous certaines conditions de valeurs d'autres tests.

Par exemple, dans la table précédente, on ne s'intéresse à l'âge que si l'ancienneté est comprise entre 10 et 20, ce qui introduit beaucoup de cases vides. On pourrait donc modéliser cette table dans l'arbre suivant :

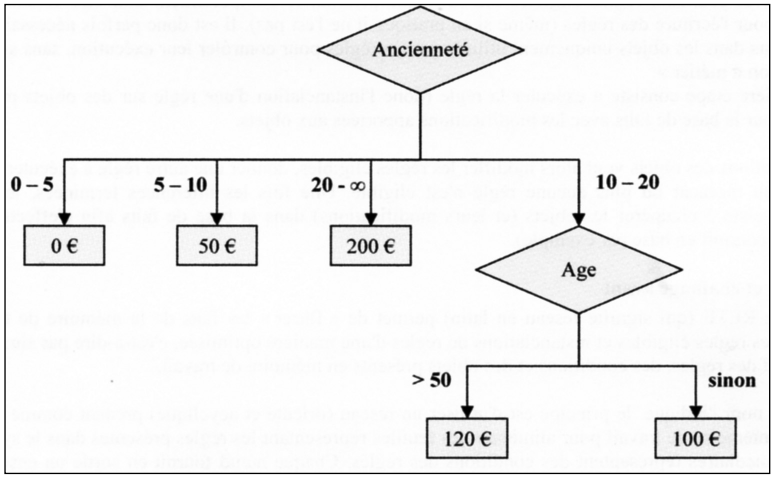

Figure 4 : Exemple d'arbre de décision

2.3 Moteur d'inférence

2.3.1 Principes

On trouve une première phase d'initialisation du moteur :

- Les règles sont compilées (pour constituer le réseau RETE servant à filtrer les règles éligibles en fonction des données)

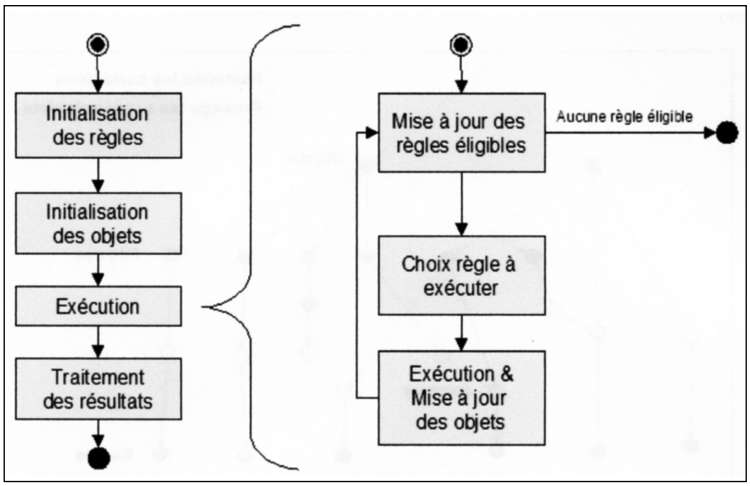

Figure 5 : Fonctionnement du moteur d'inférence

- Des objets (appelés « faits ») sont ajoutés au moteur (et servent de points d'entrée du réseau RETE).

Puis une phase d'exécution, constituée d'inférences :

- La première étape détermine les règles éligibles et les objets associés. Une règle éligible est une règle ayant des conditions vérifiables (évaluées à vrai) sur un certain jeu d'objets (t-uple) présent en mémoire de travail. Cet ensemble « règle + objets associés » est désigné par une « instanciation de règle ». On peut, en effet, considérer que la partie « action » d'une règle est une fonction prenant des paramètres, appelée sous certaines conditions et avec les paramètres récupérés dans les conditions. L'ensemble des instanciations de règles est appelé l'agenda (ou le « conflict set »).
- La deuxième étape consiste à choisir une règle à exécuter dans l'agenda. La règle choisie est la plus prioritaire. Si plusieurs règles éligibles ont la même priorité, le comportement doit être considéré comme aléatoire pour l'écriture des règles (même si en pratique il ne l'est pas). Il est donc parfois nécessaire d'ajouter des attributs dans les objets uniquement utilisés par les règles pour contrôler leur exécution, sans aucune autre signification « métier ».
- La dernière étape consiste à exécuter la règle (donc l'instanciation d'une règle sur des objets précis) et de mettre à jour la base de faits avec les modifications apportées aux objets.

Les modifications des objets vont alors modifier les règles éligibles, donner une autre règle à exécuter, et ainsi de suite jusqu'au moment où plus aucune règle n'est éligible. Une fois les inférences terminées, le traitement standard consiste à récupérer les objets (et leurs modifications) dans la base de faits afin d'effectuer d'autres traitements (commit en base par exemple).

2.3.2 RETE et chaînage avant

L'algorithme RETE (qui signifie réseau en latin) permet de « filtrer » les faits de la mémoire de travail pour déterminer les règles éligibles et instanciations de règles d'une manière optimisée, c'est-à-dire pas simplement un parcours naïf des règles, des conditions et des objets présents en mémoire de travail.

Comme son nom l'indique, le principe est d'utiliser un réseau (orienté et acyclique) prenant comme sources les objets de la mémoire de travail pour alimenter des feuilles représentant les règles présentes dans le système. Les nœuds intermédiaires représentent des conditions des règles. Chaque nœud fournit en sortie un ensemble de t-uples (de faits, c'est-à-dire d'objets de la mémoire de travail), résultant des t-uples d'entrée (du nœud) et des filtrages appliqués par le nœud.

Globalement, il existe deux types de nœuds :

- Nœuds alpha : de simple test de filtrage, analogue à une clause WHERE en SQL (test de la valeur d'un attribut par exemple)
- Nœuds bêta : des nœuds de jointure, analogue à un JOIN … ON en SQL (association de deux objets au sein du t-uple)

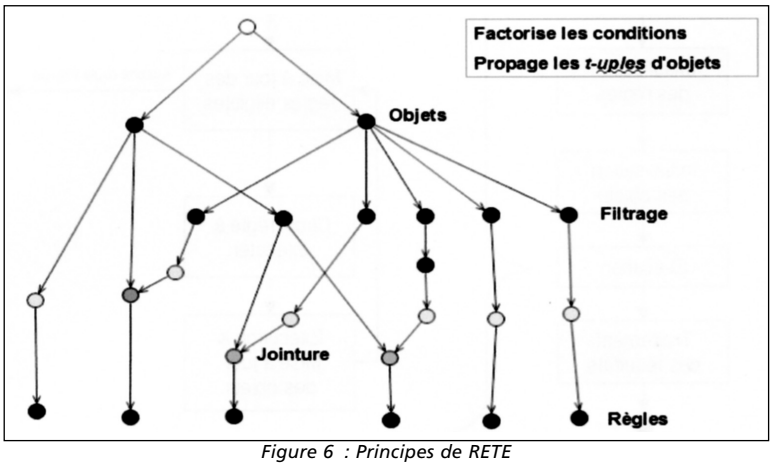

Figure 6 : Principes de RETE

Le réseau sert à factoriser les conditions des règles, et ainsi optimiser la mise à jour des règles éligibles en fonction des modifications des faits. En effet, deux règles partageant les mêmes conditions dans le même contexte (deux règles ayant leurs premières conditions en commun) vont en fait partager les mêmes nœuds dans le réseau. Le chemin vers leurs nœuds d'activation respectifs commence de la même manière, pour finir par se dissocier (sur les nœuds représentant les tests les différenciant).

Ainsi, une mise à jour de la base de faits est propagée dans ce réseau, et permet de modifier l'agenda sans avoir à parcourir l'ensemble de la base de faits pour l'ensemble des règles à chaque modification. Cette tâche de propagation est bien évidemment la plus coûteuse dans le temps global d'inférence d'un moteur, et c'est donc à ce niveau que les moteurs commerciaux apportent une grande plus-value en termes d'optimisations (en

plus des optimisations dès la construction du réseau). Par exemple, une optimisation évidente (appelée LEAPS) consiste à ne seulement calculer qu'un seul t-uple à la sortie des nœuds, et à ne générer les autres que si cela est nécessaire, par requête d'un des nœuds suivants (analogue d'un curseur en SQL).

Notons que l'utilisation la plus classique du réseau RETE est le chaînage avant, c'est-à-dire le principe décrit ci-dessus consistant à partir des faits et chercher toutes les règles à exécuter. Ce réseau (s'il dispose de mécanismes comme LEAPS, c'est-à-dire la génération d'un t-uple sur demande montante) permet également de faire fonctionner un système en chaînage arrière (limité, non récursif par exemple, le réseau étant acyclique), en partant d'une règle (donnée explicitement, qu'on nommera dans ce cas un « but ») et en cherchant à la vérifier, c'est-à-dire à trouver les t-uples vérifiant ses conditions.

2.4 Fonctionnement en mode séquentiel

La plupart des outils de BRMS disposent d'un fonctionnement en mode d'exécution « séquentiel ». Dans ce cas, il n'y a pas d'inférences et donc pas de mise à jour du réseau : le temps d'exécution est alors beaucoup plus rapide. Ce mode séquentiel est en fait un moyen d'exprimer facilement et lisiblement des boucles et des tests compliqués avec un fonctionnement impératif global. Les règles sont alors traitées en séquence :

- l'ordre d'exécution des règles est strict,
- si une règle modifie un objet qui aurait pu déclencher une autre règle déjà traitée, il n'y aura pas de chaînage.

De plus, pour une règle, les objets sont eux aussi traités en séquence : si une règle modifie un objet déjà traité, rien ne sera re-déclenché pour cet objet.

D'après les retours sur les utilisations des outils de BRMS, une grande majorité des applications en entreprises utilisent le mode séquentiel. En effet, ce mode est adapté à la plupart des applications de gestion qui veulent un comportement prédictible (une application calculant des primes doit pouvoir justifier simplement le calcul effectué).

Toutefois, le mode séquentiel n'est pas du tout adapté à des applications d'intelligence artificielle sensées modéliser un raisonnement, et qui incluent généralement :

- une remise en cause : incohérence des faits initiaux, entre eux ou par rapport à des conclusions futures,
- une production de faits déduits des faits initiaux, impliquant d'autres conclusions en chaîne,
- des déductions par omission, agrégation, simultanéité, ...

2.5 Exemples d'application des BRMS

Les principales applications des BRMS se trouvent dans l'informatique de gestion, en particulier dans les secteurs bancaire, de la finance, des assurances, et des ressources humaines. Par exemple :

- Calcul de prime
- Calcul de commissionnement
- Évaluation de risque
- Souscription de contrat
- Détection de fraude
- Validation de formulaires
- Vérification de conformité par rapport à des législations et des règlementations : vérification de la conformité réglementaire d'un planning ou d'un service d'un agent (en particulier dans le domaine des transports : conducteurs de trains, chauffeurs de bus, personnel aérien navigant), ...

3. Moteurs de règles spécifiques

3.1 Moteurs C++ ou Java appelés dans des algorithmes d'optimisation

Certains algorithmes d'optimisation et de recherche opérationnelle font appel à des moteurs de règles, par exemple afin de résoudre des problèmes de planification de personnel :

- dans le transport public (bus, tramways, métros, trains) :
  - construction des services (horaires) des agents (conducteurs de trains, métros, tramways ou bus),
  - planification nominative de ces agents,
- dans le transport aérien,
- ...

En effet, les solutions de ces différents problèmes doivent respecter des réglementations (internes à une entreprise, propres à un secteur d'activité, ...) et des législations (code du travail, ...) :

- transports publics :
  - un agent ne doit pas conduire plus de X heures dans une journée
  - un agent ne doit pas conduire plus de Y heures de manière continue,
  - un temps de pause quotidien doit être respecté,
  - si un agent conduit plus de Z heures dans une journée alors le dépassement lui sera payé en heures supplémentaires,
  - ...
- transport aérien :
  - un personnel navigant ne doit pas naviguer plus de X heures mensuellement,
  - deux rotations successives doivent être espacées d'au moins Y heures,
  - ...

Les solutions fournies par les algorithmes d'optimisation doivent donc être testées par un moteur de règles qui implémente les réglementations et les législations concernées. Or, ce type

d'algorithmes (par exemple programmation linéaire en nombres entiers et génération de colonnes, algorithme hongrois, ...) génèrent plusieurs centaines de milliers, voire des millions, de solutions-candidates. La validité de toutes ces solutions-candidates devant être évaluée, le moteur de règles doit donc être extrêmement performant d'un point de vue « temps d'exécution ». C'est pourquoi, dans ces cas particuliers où le temps de calcul est une exigence forte voire critique, ces moteurs de règles spécifiques seront plutôt codés directement dans un langage informatique de type C++ (voire Java). Un langage de définition et une Interface Homme-Machine doivent alors être développés si le projet a des objectifs d'évolutivité.

De tels moteurs de règles spécifiques, développés directement en C++ ou en Java, peuvent aussi être déployés sur un serveur et appelés en tant que service, dans une architecture de type SOA, par d'autres applications du SI (par exemple : modification d'un planning depuis une Interface Homme-Machine ou depuis un progiciel par un utilisateur « métier » et vérification automatique de la conformité de la modification effectuée).

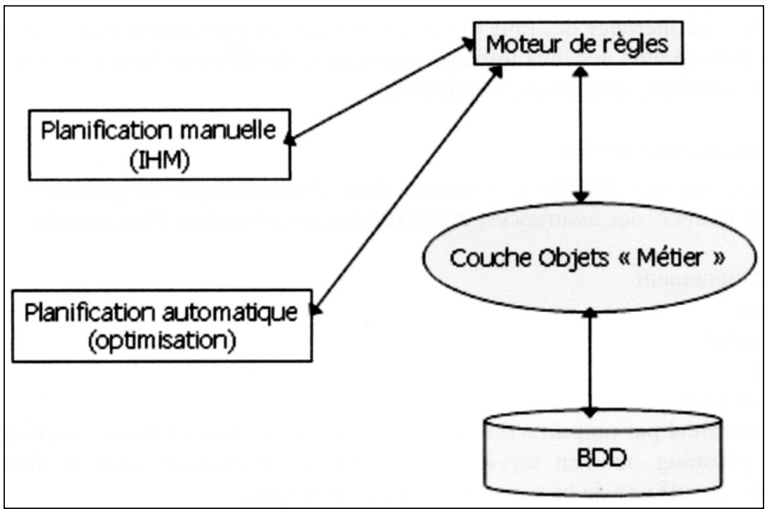

Figure 7 : Architecture intégrant des moteurs d'optimisation appelés par des algorithmes d'optimisation

3.2 Moteurs PROLOG

PROLOG est un langage de programmation logique, basé sur le calcul des prédicats de la logique du premier ordre, et sur les concepts d'unification, de récursivité et de backtracking. PROLOG peut donc servir à résoudre des problèmes logiques relatifs à une base de connaissances. Ainsi, le langage PROLOG permet d'implémenter des moteurs de règles, comme nous l'avons fait par exemple dans l'application DEMIURGE.

DEMIURGE est un logiciel d'aide à la réalisation d'études de capacité et de conception de grilles prospectives dans le domaine ferroviaire. Ce logiciel a été développé, en collaboration avec EURODECISION, par la Direction de l'Ingénierie et la Direction de l'Innovation et de la Recherche de la SNCF, et est actuellement utilisé par des experts en exploitation ferroviaire de la Direction de l'Ingénierie de la SNCF.

Lors de la réalisation d'une étude de capacité ferroviaire, les experts sont amenés à étudier plusieurs variantes, correspondant à différentes hypothèses quant à la définition des circulations (temps de parcours différents selon les matériels, utilisation d'itinéraires alternatifs, ...). Les utilisateurs doivent, pour chaque variante, effectuer les modifications de paramétrage correspondant aux hypothèses sur tous les trains concernés et sur tout le parcours de ces trains. Le réseau ferroviaire étant modélisé dans DEMIURGE sous la forme d'un graphe (nœuds et tronçons reliant ces nœuds), lorsque le nombre de nœuds augmente, le temps de saisie manuelle de ce paramétrage devient vite prohibitif. De plus, lorsque ces saisies sont nombreuses et répétitives, les erreurs deviennent fréquentes.

Afin de réduire le temps de paramétrage des variantes et d'éviter les erreurs de saisie, les utilisateurs ont souhaité disposer d'un système de paramétrage des circulations à base de règles logiques. Par exemple :

- pour tous les trains de la catégorie TER passant par le nœud A du réseau,
- les temps de parcours maximaux autorisés sont équivalents aux temps de parcours types avec une marge de 15%,
- et un itinéraire alternatif est possible au nœud A.

Nous avons développé un moteur d'inférences en PROLOG, fonctionnant en mode séquentiel et permettant de traiter de telles règles logiques de paramétrage sur la base des circulations, ainsi qu'une Interface Homme-Machine pour éditer simplement ces règles en langage naturel. De plus, une mise en correspondance entre les objets « métier » présents dans DEMIURGE et les structures de données du code PROLOG a été réalisée. Enfin, des rapports concernant l'exécution de ces règles sont fournis aux utilisateurs (erreurs, nombre d'objets modifiés, traces, ...). Ainsi, nous avons conçu et intégré dans DEMIURGE une sorte de BRMS spécifique à une problématique (figure 8).

5. Bibliographie

Systèmes de gestion de règles métier

- Pierre Bonnet, Jean-Michel Detavernier et Dominique Vauquier : *Le système d'information durable : la refonte progressive du SI avec SOA ;* Lavoisier-Hermes Science Publications, 2007
- Ronald G. Ross : *Principles of business rules approach ;* Addison-Wesley, 2003
- Tony Morgan : *Business rules and information systems: Aligning IT with business rules ;* Addison-Wesley, 2002
- Barbara von Halle : *Business rules applied: Building better systems using the business rules approach ;* Wiley, 2001

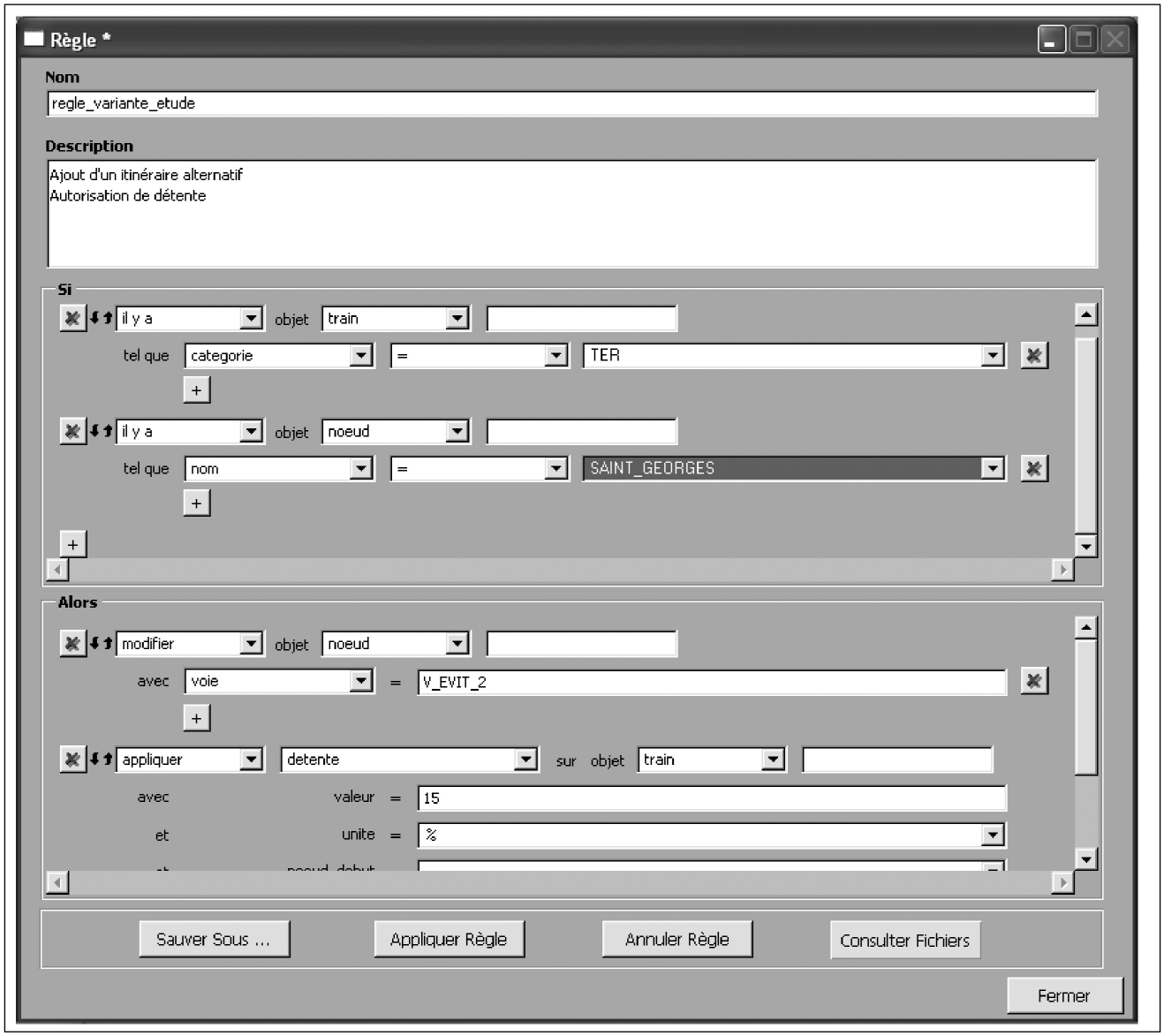

Figure 8 : Exemple d'IHM permettant d'éditer des règles dans un moteur Prolog

- Paul Browne : *JBoss Drools business rules* ; Packt Publishing, 2009
- Michal Bali : *Drools JBoss Rules 5.0 Developer's Guide* ; Packt Publishing, 2009

Intelligence artificielle

- Stuart Russel et Peter Norvig : *Intelligence artificielle* (2e édition) ; Pearson Education France, 2006

Systèmes experts et RETE

- Alain Bonnet, Jean-Paul Haton et Jean-Michel Truong : *Système experts : vers la maîtrise technique* ; InterÉditions, 1986
- Michel Le Seac'h : *Développer un système expert* ; édiTests, 1989
- Peter Jackson : *Introduction to expert systems ;* Addison-Wesley, 1999
- Charles Forgy : *On the efficient implementation of production systems*, Ph.D. Thesis, Carnegie-Mellon University, 1979
- Charles Forgy : *Rete: A fast algorithm for the many pattern/many object pattern match problem* ; Artificial Intelligence, 19, pp 17-37, 1982
- Ernest Friedman-Hill : *JESS in action: java rule-based systems* ; Manning Publications, 2003
- Joseph C. Giarratano et Gary D. Rile *: Expert systems: Principles and programming* (Fourth Edition) ; Course Technology, 2004
- Jean-Paul Delahaye : *Systèmes experts : organisation et programmation des bases de connaissances en calcul propositionnel* ; Eyrolles, 1987

PROLOG

- Jean-Paul Delahaye : *Cours de Prolog avec Turbo Prolog* ; Eyrolles, 1988
- Alain Colmerauer et Philippe Roussel : *The birth of Prolog* ; dans History of Programming Languages, rassemblé par Thomas J. Bergin et Richard G. Gibson, ACM Press/Addison-Wesley, 1996

Recherche Opérationnelle, Optimisation, Programmation Linéaire

- Christelle Guéret, Christian Prins et Marc Sevaux : *Programmation linéaire* ; Eyrolles, 2000

• Eric Jacquet-Lagrèze : *Programmation linéaire - Modélisation et mise en œuvre informatique* ; Collection P.I.Q. Poche, Economica

Sites Web

• IBM Websphere ILOG JRules : *http://www-01.ibm.com/software/integration/business-rule-management/jrules/*
• FICO Blaze Advisor : *http://www.fico.com/en/Products/DMTools/Pages/ FICO-Blaze-Advisor-System.aspx*
• JBOSS Rules – DROOLS : *http://www.jboss.com/products/platforms/brms/ http://labs.jboss.com/drools*
• Prolog :
SWI Prolog : *http://www.swi-prolog.org/*
GNU Prolog : *http://www.gprolog.org*

Les auteurs

Vincent Legendre et Gérald Petitjean sont consultants seniors au sein d'EURODECISION. Ils conçoivent des applications industrielles d'aide à la décision depuis plus de dix ans.

Thierry Lepatre est expert « Exploitation Ferroviaire et Études de Capacité » au sein de la Direction de l'Ingénierie de la SNCF et chef de projet « métier » de l'application DEMIURGE.

EURODECISION

La mission d'EURODECISION est d'apporter aux entreprises une offre globale (conseil, logiciel, service) pour réaliser des solutions d'aide à la décision dans les domaines de la logistique, de la production, des ressources humaines et de la conception de produits, en intégrant des technologies avancées issues de la Recherche Opérationnelle, de l'Optimisation, des Mathématiques Appliquées et de l'Intelligence Artificielle (programmation linéaire, programmation par contraintes, méta-heuristiques, prévision, systèmes à base de règles, calcul parallèle et HPC) :

• *Logistique :* localisation et dimensionnement des unités de production, des hubs et des entrepôts, optimisation des flux, dimensionnement des flottes (camions, wagons, navires, avions…), gestion des stocks, affectation des zones de chalandise, optimisation des plans de transport (approvisionnement, distribution, messagerie, multi-modalité …)…
• *Production :* dimensionnement des unités de production, planification et ordonnancement de la production, planification de la maintenance, planification de services (construction des horaires d'avions, de trains…)…
• *Planification des Ressources Humaines :* planification annuelle, dimensionnement des effectifs, construction optimisée d'horaires de personnels pour couvrir une charge (journalière ou hebdomadaire), construction optimisée d'horaires de personnels (chauffeurs de bus, métros, tramways, …) par assemblage de tâches, construction de roulements et de tableaux de services cycliques, planification nominative et affectation de personnels, …
• *Conception :* méthodes et outils d'optimisation de la conception de produits s'appuyant sur des simulations numériques, afin de trouver les meilleurs compromis possibles au regard du cahier des charges.

EURODECISION a développé un savoir-faire reconnu par des grandes entreprises telles que : Aéroports de Paris, Airbus, Air Liquide, Air France, Alcan, Brinks, Decathlon, DGA, EDF, Italcementi, Lactalis, La Poste, RATP, RTM, Renault, SNCF, Total, Totalgaz, VEOLIA Transport, ...

En investissant 15% de son chiffre d'affaires et en participant à des projets collaboratifs français et européens EURODECISION place la R&D au cœur de sa stratégie. L'entreprise est membre de plusieurs pôles de compétitivité (SYSTEM@TIC, ADVANCITY, NOV@LOG) et associations scientifiques.

Renseignements : *http://www.eurodecision.fr/*