# Le contrôle dans Shal

## Synthèse

Cette note vise à décrire le contrôle dans Shal. Shal est une extension de Boojum/Genesia II, qui intègre différents éléments, dont un contrôle explicite par plans et méta-règles. Il s'avère que le manque de contrôle de l'inférence dans Boojum ou Genesia II est certainement ce qui a le plus manqué aux programmeurs de règles dans ces langages. Il y fallait "jouer" avec la stratégie fixe du moteur en ajoutant aux règles des prémisses et conséquents "fictifs" par rapport à leur véritable contenu, mais visant à inhiber ou réveiller leur déclenchement. Or, les solutions adoptées dans Shal améliorent considérablement la situation par rapport à ses ancêtres (même si elles ne résolvent pas tout).

Cette note vise à permettre aux programmeurs de règles d'accéder à Shal, du moins sous son aspect contrôle. Leur productivité en sera probablement fortement accrue, ainsi que la déclarativité de leurs règles. Sont visés *en particulier* les intervenants du PPRD Descartes, qui programment encore actuellement les composants du synthétiseur de programme en Genesia II.

Les trois composants du contrôle d'inférence dans Shal sont : le regroupement des règles en paquets, appelés expertises; des plans, fournis par l'utilisateur, qui indiquent une suite d'actions conditionnelles pour atteindre un but; enfin, un ensemble (normalement fixe et indépendant de l'application) de méta-règles, qui exécutent un plan pour atteindre un but. Il ne suffit donc plus de fournir à Shal une base de règles (regroupées en expertises) et une base de faits initiale pour qu'il commence l'inférence, il faut aussi lui fournir un but à atteindre. Cette note précise la définition et l'utilisation que l'on peut faire de ces nouveaux moyens de contrôle. Une description de ce comment ce contrôle est mis en œuvre est également tentée, sans toutefois rentrer dans tous les détails techniques.

On mentionnera enfin les extensions possibles de ces éléments de contrôle, celles-ci étant entreprises selon les besoins.

# 1 Motivation

Le langage de départ de Shal est à peu près celui de Boojum [Dormoy 87], où la résolution de conflits[1] est élémentaire. La stratégie de Boojum est de sélectionner la première règle déclenchable (première dans l'ordre où elles sont fournies) et de saturer les déclenchements de cette règle. La raison pour laquelle nous avions adopté cette stratégie simple est qu'il n'y a pas de "bonne" stratégie de résolution de conflits indépendante de la base de connaissances à laquelle elle s'applique. Sa simplicité est aussi un avantage pour sa mise en œuvre informatique et pour sa compréhension par le concepteur de bases de règles.

Cependant, il est bien connu que l'existence d'une stratégie de résolution de conflits fixe pose de graves problèmes au concepteur de bases de règles. Celui-ci est contraint de la connaître dans le détail, et souvent de "tricher", de la "tromper", en introduisant notamment des "faits de contrôle" pour que les règles s'appliquent dans l'ordre qu'il désire. La situation n'a pas changé avec Genesia II.

Jusqu'à présent, les concepteurs de bases de règles en Boojum ou Genesia II ont vécu avec ces problèmes. Dans la construction de Shal, la simple survie s'est avérée impossible pour deux raisons. Premièrement, "flaguer" toutes les règles d'un grand système s'avère vite incompréhensible. Deuxièmement, ce qui est vrai pour le concepteur l'est encore plus pour Shal. Shal tente en effet de "comprendre" la base de connaissances qui lui est fournie. Les connaissances pour retrouver le contrôle implicite exprimé dans les faits de contrôle seraient, si nous avions tenté de les écrire, sérieusement sophistiquées ! Nous avons donc choisi une voie plus simple : donner explicitement le contrôle. Cela a le triple avantage de simplifier la vie du concepteur, de simplifier celle de l'interpréteur, qui ne sera plus encombré par les faits de contrôle, et de simplifier les connaissances analysant le comportement d'une base de connaissances. En fait, ces dernières connaissances disposent maintenant d'une meilleure information qu'auparavant.

Pour autant, si c'est la conjonction de ces motivations qui a emporté la conviction (pour se décider à mettre en œuvre ce contrôle), sa réalisation actuelle serait fort utile à n'importe quel concepteur de base de connaissances. Qu'il en juge par ce qui suit.

# 2 Présentation

Cette note vise avant tout à être utilitaire. Cependant, nous n'avons pu résister à la tentation de décrire "comment çà marche". Ce qui a abouti aux trois parties qui suivent : d'abord une description conceptuelle des éléments du contrôle (sans description des moyens syntaxiques de leur mise en œuvre); puis une description des

[1] La résolution de conflits est l'opération qui consiste à choisir avant chaque déclenchement une parmi toutes les instanciations valides à ce moment .

mécanismes internes "exécutant" le contrôle défini par l'utilisateur (et quelques remarques sur son efficacité); enfin, la description syntaxique du contrôle ("comment le programmer").

# 3 Description du contrôle

Ce paragraphe offre un panorama général du contrôle dans Shal. Le lecteur désireux de le lire en suivant un exemple peut se reporter au § 6.2.

## 3.1 Buts et plans

Nous avons choisi dans un premier temps un moyen d'expression du contrôle simple, mais en tentant de prévoir les extensions futures. Tout d'abord, les règles sont regroupées par le concepteur en expertises, qui ne sont rien d'autre que des paquets de règles. Ensuite, l'utilisateur fournit des buts potentiels que le système peut avoir à résoudre, et des plans pour résoudre ces buts. Les buts ont une forme purement propositionnelle (ils ne mentionnent pas de variables). Ils utilisent des identificateurs qui peuvent n'avoir rien de commun avec les objets ou relations utilisés dans la base de connaissances auxquels ils se réfèrent. Le seul lien qu'ils auront avec celle-ci sera via les plans qui permettent de les atteindre. Des exemples de buts sont CompilerBaseDeConnaissances, SynthétiserTraducteur, MonBut, etc (mais aussi, nous le verrons dans la description de la mise en œuvre du contrôle, Planifier).

Les plans sont des successions d'actions conditionnelles à effectuer. A chaque but est associé un plan, actuellement unique[2].

## 3.2 Actions et transitions

Les actions se résument actuellement à exécuter une expertise, ou un paquet d'expertises (en vrac), à atteindre un but, ou à exécuter un sous-programme du moteur[3]. Les plans se matérialisent en une série de transitions entre actions.

Dans un plan donné, une action A peut transiter vers d'autres actions A' de cinq manières différentes[4]. Il y a tout d'abord les transitions inconditionnelles, A --> A'. Dans ce cas, l'exécution de l'action A doit être systématiquement suivie de l'exécution de l'action A'. Il y a ensuite quatre sortes de transitions conditionnelles, A d--> A' et A nd--> A' si A est une exécution d'expertise, et A a--> A' et A na--> A' si A est une résolution de sous-but. Si on a une transition A d--> A' (resp. A a--> A'), alors l'exécution de l'action A doit être suivie de l'exécution de l'action A' si au moins une règle de l'expertise exécutée dans A s'est déclenchée (resp. si le sous-but a été atteint). A nd--> A' (resp. A na--> A') a l'effet inverse: A' sera exécutée si aucune

---

[2] Il n'y a pas de méta-règles de choix selon le contexte d'un plan entre plusieurs pouvant atteindre un but, mais cette extension est envisageable.

[3] Quoique cette dernière possibilité soit d'utilisation délicate pour un utilisateur non averti. Nous n'en donnons pas une description complète.

[4] Plus quelques autres que nous mentionnerons au passage.

règle de l'expertise exécutée dans A ne s'est déclenchée (resp. si le sous-but n'a pas été atteint).

L'exécution d'une expertise (qui regroupe plusieurs règles) suit par défaut de manière interne la stratégie de résolution de conflits de Boojum. Cependant, une action d'exécution d'expertise peut être accompagnée d'ordres qui changent cette stratégie. On peut ainsi limiter l'inférence aux instanciations qui ont "réveillé" les règles de l'expertise depuis leur dernier examen (à peu près les "nouvelles" instanciations); on peut borner supérieurement le nombre de déclenchements (en particulier par 1); on peut ne compter comme déclenchements que ceux modifiant la base de faits ou ayant un effet de bord[5]; ou bien une combinaison de ces possibilités.

L'action qui consiste à atteindre un sous-but va simplement provoquer l'exécution du plan associé. Lorsque ce "sous-plan" aura achevé son exécution, le cours normal du plan "appelant" reprendra. Cependant, ce "sous-plan" renvoie au plan appelant une information disant si le sous-but a été atteint ou pas, information qui sert à décider de la transition suivante.

### 3.3 Quand un but est-il satisfait ?

Par contrecoup, tout plan mentionne des conditions qui indiquent quand le but pour lequel il est conçu a été satisfait ou non. Cela peut être qu'une certaine action A a été exécutée (sans condition sur le résultat de cette action). On utilise ce genre de chose lorsque A est située sur une branche du plan où l'on est sûr du résultat. Cela peut aussi être l'exécution d'une certaine action, mais cette fois en tenant compte de son résultat. Si l'action consiste à exécuter une expertise, la condition de satisfaction peut mentionner une contrainte sur le nombre de déclenchements de ses règles. Si l'action consiste en un appel de sous-but, la satisfaction du sous-but peut être considérée. Le succès d'un plan (et donc du but origine) se limitent actuellement à ces possibilités. D'autres pourraient être ajoutées.

### 3.4 Quand un plan est-il terminé ?

Enfin, nous avons parlé de terminaison d'un plan. Un plan est terminé lorsqu'aucune transition valide n'existe. Il est également terminé lorsque le but a été atteint (quoique cela ne soit vrai que par défaut, car on peut définir une condition de succès d'un plan en forçant le plan à continuer).

Tous les identificateurs utilisés pour la spécification des buts et des plans sont libres, et a priori sans rapport avec ceux utilisés dans la base de connaissances. Seule exception : les expertises ont un nom, et l'appel d'une expertise se fait par son nom.

### 3.5 Quelques utilisations des plans

---

[5] Nous entendons par "effet de bord" une action qui ne modifie pas directement la base de faits, mais dont la répétition peut être indispensable, comme une entrée-sortie ou l'appel d'un programme externe.

Les "plans" sont donc très simples. Premièrement, ils sont fournis par le concepteur, et non trouvés par le système. Ils ressemblent actuellement plutôt à un langage de commande. Malgré tout, ils permettent d'exprimer le contrôle plus facilement, et d'introduire de nouveaux éléments de contrôle.

On peut ainsi vouloir orienter l'inférence en certains moments cruciaux selon l'état de la base de faits. On peut par exemple imaginer des contraintes qui, si elles sont vérifiées, devraient aiguiller l'inférence dans un sens, et sinon dans un autre. Ces contraintes peuvent être mises en œuvre comme des expertises qui contiendront des règles (souvent une seule) *sans conséquent*. Exécuter une telle expertise permet au plan de contrôle de savoir si la partie gauche d'une de ces règles est satisfaite (via le nombre de "déclenchements"), et d'agir en conséquence.

Autre exemple, il est possible d'introduire des circuits dans les plans, c'est-à-dire ... des boucles. Il est aussi possible d'écrire des expertises pour chaque but potentiel du système qui regardent si le but est satisfait, ou s'il faut abandonner le plan. De telles expertises fournissent l'expression des conditions de satisfaction des buts ou des conditions d'arrêt. Enfin, on peut écrire des expertises qui regardent si le plan se déroule correctement, et qui peuvent donc récupérer des erreurs.

Enfin, nous avons aussi introduit le parallélisme entre les actions, c'est-à-dire qu'une action peut être suivie de plusieurs actions, et que plusieurs actions peuvent être suivies d'une action unique (paquets d'expertises). Cela ne change rien quant à la logique des plans, mais apporte une source de connaissances supplémentaire pour la compréhension du comportement et pour le debugging, car des actions en parallèles sont réputées commutatives. Cela rend aussi l'écriture des plans un peu plus "déclarative", en n'imposant pas de séquencement des actions vide de sens.

# 4 Comment ça marche ?

## 4.1 Expertises d'exécution des plans

Venons-en maintenant à la manière dont sont exécutés les plans. Normalement, un utilisateur qui se satisferait des moyens de contrôle que Shal est actuellement capable de lui fournir n'a pas besoin de connaître ce qui suit. Mais nous savons bien que l'utilisateur est toujours curieux, et quelquefois insatisfait. Aussi s'avère-t-il qu'il aura les moyens d'accroître *lui-même* les possibilités de contrôle de Shal, du moins dans certaines limites. Le contrôle est en effet exécuté par des méta-règles, qui n'ont rien de spécial par rapport à des règles ordinaires, sinon qu'elles agissent sur le contrôle au lieu du problème de base.

Les plans sont exécutés par des expertises spéciales, des expertises d'exécution de plans. Ces expertises sont exécutées par un plan spécial qui tente de satisfaire un but, qui est de satisfaire le but courant du système de base. Ce but spécial s'appelle Planifier, et le plan spécial associé au but Planifier s'appelle PlanPourPlanifier. Les méta-règles de contrôle sont donc elles-mêmes contrôlées, comme de vulgaires règles ordinaires qu'elles sont, par un plan tentant de satisfaire un but.

Qui exécute ce plan ? Justement les expertises d'exécution de plans, contrôlées par ce même plan. Nous sommes partis dans les tours infinies de niveaux méta de H. Dreyfus …

### 4.2 Bases de faits méta

Pour résoudre ce problème, l'architecture du système a été modifiée et est maintenant la suivante. Le système ne possède toujours qu'un unique ensemble d'expertises, mais ces expertises pourront être utilisées à n'importe quel niveau méta. Par contre, il n'y a plus une mémoire de travail (base de faits) unique, mais *une mémoire de travail par niveau méta "actif"*. Dans la mémoire de travail "du bas", disons celle de niveau 0, on met les faits ordinairement mis dans la mémoire de travail, c'est-à-dire ceux qui représentent les éléments du problème posé. Dans la mémoire de travail de niveau 1, on met le but assigné au niveau de base, et tous les plans. Dans les mémoires de travail de niveau supérieur, on met le but du niveau juste inférieur, et également tous les plans. Il est clair que l'on retrouve très rapidement des buts similaires. Les mémoires de travail des niveaux 1, 2, 3, … se rempliront aussi au fur et à mesure des faits permettant de contrôler l'exécution des différents plans.

Prenons un exemple (cf. Figure 1). Supposons que le but du niveau de base soit MonBut. Ce but est exprimé au niveau 1, et nous voudrions que les expertises d'exécution de plans s'exécutent au niveau 1 (c'est-à-dire sur cette mémoire de travail de niveau 1) pour qu'elles retrouvent le PlanPourMonBut, et l'exécutent. Pour cela, il faut mettre au niveau 2 le but Planifier, de manière à ce que les expertises d'exécution de plans s'exécutent au niveau 2 pour exécuter ces mêmes expertises au niveau 1 selon le plan PlanPourPlanifier. Attention ! Ce sont bien les mêmes expertises d'exécution de plans qui s'exécutent aux niveaux 1 et 2, selon le même plan PlanPourPlanifier, mais elles n'exécutent pas le même plan. Au niveau 1, le plan exécuté est PlanPourMonBut, et au niveau 2 PlanPourPlanifier. Comment maintenant les expertises d'exécution de plan vont-elles être exécutées au niveau 2 ? On pourrait les faire exécuter par "elles-mêmes", prises au niveau 3, avec exactement le même but qu'au niveau 2. Mais il est clair que cela est inutile, car ces expertises vont faire (du moins dans l'état actuel du système) la même chose aux niveaux 2 et 3 - sauf que ce sera à des moments différents. Par ailleurs, il faut arrêter l'ascension dans les niveaux méta. Cela est possible en *procéduralisant* l'expertise d'exécution de plans appliquée au PlanPourPlanifier. Il suffit donc de disposer d'un programme qui "soit" l'exécution de ces expertises pour ce plan, et de le laisser s'exécuter au niveau méta ad hoc.

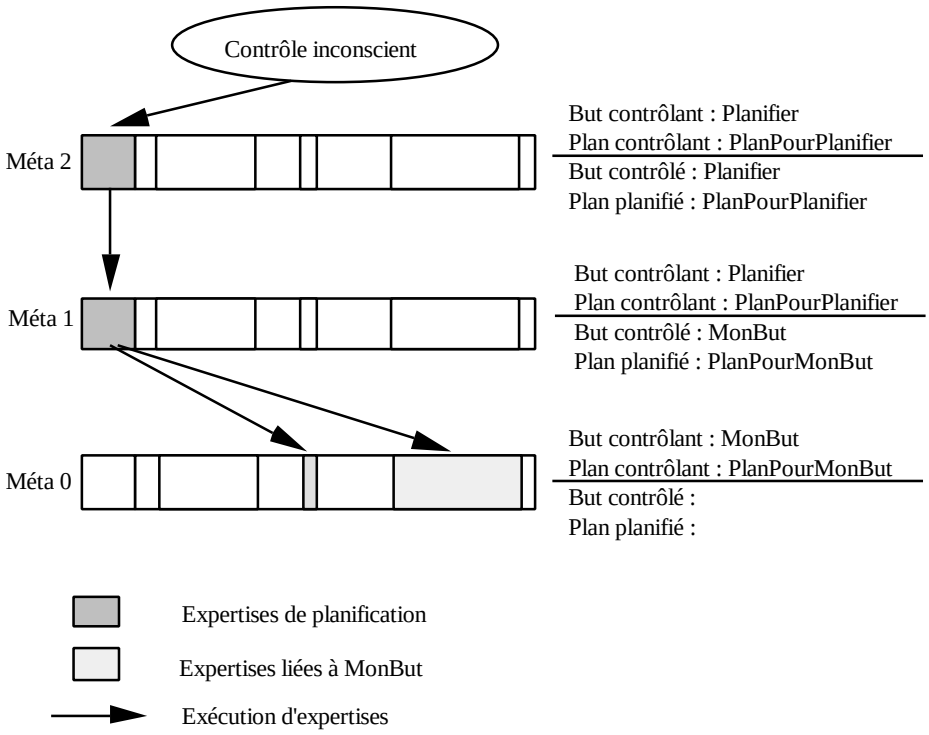


*Figure 1 : Le contrôle dans Shal*

### 4.3 Le contrôle procédural

Donc, lorsque nous disions que le contrôle était rendu déclaratif (contrôle par méta-règles), le lecteur pourrait croire que nous mentions un peu, car *il faut quelque part un programme "normal"*. Et pourtant, nous ne mentions pas.

Cette planification procéduralisée est effectivement un programme, mais nous ne l'avons pas écrit. Il est synthétisé par des expertises, dites de "synthèse du contrôle procédural", qui ne font que traduire en programme procédural la suite d'actions que donneraient les expertises d'exécution de plans lorsqu'elles sont appliquées au but Planifier en suivant le plan PlanPourPlanifier.

Ce qui précède montre que l'on peut mettre cette expertise procéduralisée, appelons-là *contrôle inconscient*, au niveau 2. Cependant, nous la mettons quelquefois au niveau 3. Cela peut sembler inutile, puisqu'elle y effectue à ce niveau exactement les mêmes actions que les expertises d'exécution de plans qu'elle contrôle au niveau 2. Cependant, des problèmes peuvent se poser lorsque l'on modifie les expertises d'exécution de plans et conséquemment celles de synthèse du contrôle procédural. Nous ne rentrerons pas plus loin dans cette discussion.

Nous avons décrit comment les mémoires de travail de chaque niveau méta sont initialisées (buts et plans pour les niveaux supérieurs). Au cours de l'inférence, ces mémoires de travail changent, et nous allons examiner ce qu'elles contiennent aux

niveaux supérieurs. Elles contiennent tout d'abord la trace d'exécution des expertises du niveau inférieur. Celle-ci consiste en la liste (ou le graphe sans circuit) des actions exécutées. Sont également présentes des informations permettant de savoir quel plan est en cours d'exécution, où on en est dans le plan, les sous-buts en examen, etc, bref ce qui correspondrait à la pile d'exécution dans un langage classique.

D'un autre côté, des informations sont nécessaires pour savoir quelle transition appliquer ensuite. Pour cela, il faut avoir accès à certaines informations *relatives à l'exécution des expertises du niveau inférieur*, et les représenter dans la mémoire de travail. Nous avons choisi dans la version actuelle de Shal de modifier le moteur pour mémoriser ces informations et les transférer dans la mémoire de travail du niveau méta ad hoc. Elles consistent, pour chaque expertise exécutée, en le nombre de déclenchement de règles de cet expertise dans l'appel (le fait qu'un sous-but est atteint est directement géré par les expertises, puisque cela n'a jamais été programmé dans le moteur). Il serait envisageable d'introduire d'autres "méta-prédicats" similaires (qui peuvent d'ailleurs être écrits comme des "fonctions externes", dont la programmation est accessible à n'importe quel utilisateur !).

### 4.4 Efficacité du méta-contrôle

Une croyance fortement enracinée est que "le méta, c'est inefficace". Nous devons avouer que nous l'avons partagée, au moins sous certains aspects. Aussi avons-nous été surpris par les résultats du méta-contrôle de Shal.

Dans Shal, le "méta" ne fait pas perdre de temps, il en fait le plus souvent gagner. Pourquoi ? Dans nos premières expériences de mise au point du méta-contrôle, nous avions pris pour exemples des bases de connaissances au niveau 0 très petites. Le résultat était rageant : pour chaque déclenchement de règle du niveau 0 (c'est-à-dire, le travail "utile"), il fallait 10 déclenchements au niveau méta, et 100 au niveau méta-méta. Le résultat était donc une dégradation des performances d'un facteur 10 ou 100 ! Puis nous avons pris des exemples plus substantiels (Shal lui-même). Le rapport était alors très différent. Pour 100 à 1000 déclenchements de règles du niveau de base, il n'y avait plus que 10 déclenchements au niveau méta. En définitive, on ne "perdait" que 1 à 10% du nombre d'inférences. La raison de la différence avec les exemples simples est que, lorsqu'une expertise du niveau de base est exécutée, c'est pour faire quelque chose de substantiel. Dans les exemples simples, l'exécution d'une expertise du niveau de base conduisait à 0 ou 1 déclenchement de règle, dans l'exemple complexe, le même type d'exécution conduisait à des dizaines ou des centaines de déclenchements. La proportion de temps perdu par le "méta" devenait donc faible.

D'un autre côté, ce temps perdu est largement compensé par un autre phénomène. Au lieu d'avoir à chercher à chaque cycle la règle à examiner parmi toutes les règles, le moteur d'inférence est d'emblée focalisé sur l'expertise à exécuter. D'autres tâches du moteur en sont également allégées. Il s'avère que ce gain de temps compense largement celui perdu par le "méta". Nous avons eu des cas où l'exécution a été trois fois plus rapide avec le "méta" que sans.

### 4.5 "Philosophie" du méta-contrôle

Pourquoi avons-nous introduit ce type de méta-contrôle ? Beaucoup de travaux ont déjà été consacrés au méta-contrôle, depuis TEIREISIAS [Davis 82] jusqu'à MACISTE [Pitrat 84-90] en passant par SNARK [Vialatte 85], [Laurière 86] et PRODIGY [Minton 89]. Ce que nous n'avons pas voulu reproduire, c'est le contrôle procédural par "appel de programme", même si actuellement ce que nous avons fait y ressemble. Shal a une liberté potentielle de son comportement. Au lieu de contenir des ordres du type "faire ça", il commence par se dire "quel est mon but ?" puis "comment vais-je y aboutir ?". La différence avec l'appel de programme sera nette lorsque le système sera capable de trouver lui-même les plans.

# 5 Utilisation du contrôle : niveaux méta et expertises

## 5.1 Spécification du niveau méta des faits

Comme dit précédemment, le contrôle (buts, plans, actions, transitions, conditions d'arrêt et de succès) est représenté en faits. Plus précisément, les éléments de contrôle se rapportant à un niveau n sont représentés dans la base de faits du niveau n+1.

Dans Shal, la base de faits a la même syntaxe que dans Boojum ou Genesia II, sinon qu'il faut préciser le niveau méta d'un fait. Cela est rendu possible par l'introduction d'un nouveau mot-clé, `meta`, suivi d'un entier naturel (0, 1, 2, …) ou -1. Les faits qui suivent une "déclaration"

```
meta 1
```

seront tous placés au niveau 1, jusqu'à ce qu'une nouvelle déclaration soit rencontrée.

Par exemple, dans la base de faits suivante

```
meta 0

Toto EstLePereDe Lulu
Lulu EstLePereDe Zonzon

meta 1

MonBut But Courant
```

les deux premiers faits sont au niveau 0, le dernier fait au niveau 1.

On autorise également l'écriture

```
meta -1
```

qui signifie que tous les faits qui suivent (jusqu'à la prochaine déclaration) sont relatifs à tous les niveaux méta. En général, on considérera que les plans sont ainsi placés à tous les niveaux, même s'ils ne sont en fait utilisés qu'à un niveau connu (car, dans Shal, selon le but courant, un même fait peut être utilisé à divers niveaux; ainsi,

lorsqu'on compile le plan PlanPourPlanifier, celui-ci est utilisé aux niveaux 0 et 1, et éventuellement 2).

Les déclarations de niveau méta peuvent être placées n'importe où dans un fichier de faits, et dans n'importe quel ordre, et plusieurs déclarations au même niveau peuvent être utilisées. Enfin, tant qu'aucune déclaration explicite de niveau méta n'est rencontrée, le niveau par défaut est 0.

### 5.2 Inférence par niveau méta

Quant aux règles, elles travaillent au niveau *fixé à un instant donné par le moteur*. Il n'y a donc pas besoin d'indication de niveau dans les prémisses ou conclusions de règles; en particulier, une règle en cours de déclenchement au niveau n ne peut être satisfaite que par des faits du niveau n, et déduire des faits à ce même niveau.

A priori, *toutes les règles et expertises sont exécutables à n'importe quel niveau méta*. C'est le déroulement des plans, ainsi que l'initialisation du contrôle (en général par exécution du contrôle procédural au niveau 2), qui fixent à quel(s) niveau(x) une expertise donnée va travailler.

### 5.3 Expertises

Les règles Shal ont à peu près la même syntaxe qu'en Boojum. La différence majeure est l'introduction des expertises. Une expertise est tout simplement une suite connexe de règles dans le fichier de règles qui débute à une déclaration

```
expertise yyy
```

où yyy est un nom donné à l'expertise (un objet Shal, en général un simple identificateur), et se finit à la déclaration d'expertise suivante ou à la fin du fichier. Entre deux déclarations d'expertises, on écrit des règles normales.

### 5.4 Fichiers "include"

La multiplication des expertises et des bases de faits nous a conduit a introduire une facilité élémentaire, l'utilisation de fichiers *include*. Cela se fait simplement par l'ordre

```
%include nomfic
```

où nomfic est un nom de fichier.

**Conseil :** Lorsque l'on utilise des fichiers include pour les faits, il est fortement conseillé de commencer le fichier par une déclaration de niveau méta (car sinon, les faits vont être placés au même niveau que les derniers faits du fichier lu juste précédemment).

### 5.5 Initialisation et passage de l'inférence entre niveaux méta

Ce sous-paragraphe ne fait que compléter la description de la mise en œuvre du contrôle par buts, plans et méta-règles. Sa lecture n'est utile au lecteur que s'il désire *modifier* les méta-règles générales d'exécution des plans.

Lorsqu'il démarre, le moteur choisit un niveau initial (en général 2), et installe un agenda (qui fonctionne actuellement comme une pile, i.e. en LIFO) en l'initialisant à une tâche unique (en général l'exécution du contrôle procédural à ce niveau 2). En pratique, donc, cette première tâche va exécuter le plan PlanPourPlanifier au niveau 1. Cela va provoquer le choix et l'exécution du plan selon le but spécifié pour le niveau de base par l'utilisateur.

Les changements de niveau se font lorsqu'une règle est déclenchée et contient en conséquent un ordre

executer(xxx)[6]

où xxx est un nom d'expertise, ou une variable instanciée par un nom d'expertise. Le moteur installe alors en tête de l'agenda l'exécution de l'expertise considérée au niveau juste inférieur au niveau courant. Le contrôle procédural contient aussi des ordres de même type (qui sont en fait menés à bien par une procédure du moteur). En fait, techniquement parlant, cet ordre empile trois tâches dans l'agenda : appel de l'expertise au niveau inférieur, rapatriement de la trace de cette exécution à la base de faits du niveau supérieur, et retour au niveau méta supérieur[7]. Après l'exécution de l'expertise du niveau inférieur, le contrôle est donc rendu au niveau appelant.

Il n'y a pas actuellement de moyen de "monter" d'un niveau méta. Cela serait en particulier utile pour faire de l'espionnage ou de l'observation (auquel cas la "montée" devrait être faite sur interruption, et non décidée par le niveau inférieur). La raison en est tout simplement que nous ne nous sommes pas encore intéressé sérieusement à ce type de méta-connaissance.

## 6 Représentation en faits des buts et plans

Nous en venons maintenant à la représentation en faits des buts et des plans.

Nous donnons dans ce qui suit une liste de schémas de faits, avec leur signification. Le lecteur saura y reconnaître tous les éléments nécessaires à la description de son contrôle. Pour s'y retrouver, le lecteur pourra aussi se reporter à l'exemple du §6.2

---

[6] Plus généralement, l'ordre a la forme
executer(xxx,yyy)
où xxx est le nom de l'expertise (resp. de la procédure du moteur) appelée et yyy sa stratégie d'exécution (resp. la liste de ses paramètres). Par défaut, yyy est 0, la stratégie de Boojum (resp. la liste vide).

[7] Le lecteur familier avec la mise en œuvre d'un interpréteur reconnaîtra dans l'agenda les analogues de la pile, du contexte et de la continuation.

(ou commencer par là s'il a l'habitude de la représentation en faits de structures informatiques), voire à la représentation du plan pour exécuter un plan au §6.3. Toutefois, l'exemple ne recoupe pas toutes les possibilités, seul ce qui suit est exhaustif (du moins dans la version actuelle de Shal). Enfin, le lecteur désireux de modifier le contrôle fera bien de demander conseil à l'auteur de cette note.

Enfin, le §6.4 décrit des possibilités de contrôle supplémentaires (utilisées essentiellement dans la mise en œuvre de Shal lui-même, et que nous insérons ici par souci d'exhaustivité).

### 6.1 Schéma de faits

**Notations :** On note en caractères gras les classes d'objets, en caractères normaux les objets (relations et autres) constants. Les schémas de faits sont suivis, lorsque nécessaire, d'un commentaire en italique.

Avant les plans, nous décrivons comment on représente le but à faire atteindre par le système (pour un but du niveau 0, le fait suivant est donc mis au niveau méta 1).

**but** But Courant

Voici maintenant la description des plans.

(**but** PlanPotentielPourBut **plan**) ConditionPlanButContexte nil
***plan*** *est un plan pour* ***but*** *(actuellement un seul plan par but). La condition est actuellement toujours* ***nil*** *(devrait servir dans des extensions ultérieures).*

**plan** EtapeDePlan **objet** : **etape**
*Ces triplets représentent les étapes des plans. L'objet* ***objet*** *est n'importe quoi distinguant* ***etape*** *des autres étapes du plan.*

**but** Atteindre But : **action_but**
**expertise** Executer Expertise : **action_exp**
**paq_exp** Executer PaquetDExpertises : **action_paq**
*Les trois types d'actions. Les fonctions compilées dans le moteur sont considérées comme des expertises (le moteur sait les reconnaître).*

**paq_exp** Element PaquetDExpertises
**expertise** Element **paq_exp**
*Description d'un paquet d'expertises (identique à la description standard d'un ensemble).*

**etape** ActionPlan **action**
*Description de l'action* ***action*** *de l'étape* ***etape****. Attention ! Des étapes distinctes peuvent correspondre à la même action, mais en étant situées à des endroits différents du plan.*

**etape** ActionInitialePlan **plan**

*Le plan **plan** débute par l'étape **etape**.*

**but** SousButAtteint Oui : **cond_succes1**
**but** SousButAtteint Non : **cond_succes2**
**but** SousButExamine Oui : **cond_succes3**
**expertise** ExpertiseDeclenchee Oui : **cond_succes4**
**expertise** ExpertiseDeclenchee Non : **cond_succes5**
**expertise** ExpertiseDeclenchee AuMoinsUneFois : **cond_succes6**
**expertise** ExpertiseExaminee Oui : **cond_succes7**
**expertise** ExpertiseExaminee AuMoinsUneFois : **cond_succes8**
**expertise** PaquetDExpertisesDeclenche Oui : **cond_succes9**
**expertise** PaquetDExpertisesDeclenche Non : **cond_succes10**
**expertise** PaquetDExpertisesDeclenche AuMoinsUneFois : **cond_succes11**
**expertise** PaquetDExpertisesExamine Oui : **cond_succes12**
**expertise** PaquetDExpertisesExamine AuMoinsUneFois : **cond_succes13**
*Les conditions de succès d'un plan. Celles-ci portent sur des actions, pas des étapes. Une expertise est* déclenchée *dès qu'une règle s'est déclenchée. Un paquet d'expertises est* déclenché *dès qu'une de ses expertises est déclenchée. **AuMoinsUneFois** signifie que le but est atteint, mais que le plan continue.* ...Examine(e) Oui *signifie que le déroulement du plan a conduit à passer par l'exécution de l'action en question.*

**cond_succes** ConditionButAtteint **plan**
*Les conditions de succès d'un plan (le plan s'arrête dès qu'une condition de succès est remplie).*

**etape** --> **etape**
**etape** d--> **etape**
**etape** nd--> **etape**
**etape** a--> **etape**
**etape** na--> **etape**
**etape** sd--> **etape**
**etape** snd--> **etape**
*Les transitions des plans (d, nd : l'étape source est une exécution d'expertise; a, na : sous-but; sd, snd : exécution d'un paquet d'expertises).*

**{**0**,**-1**,**-10**,**-11**,**-100**,**-101**,**-110**,**-111**}** Element
StrategieDExecutionDExpertiseProcedurale : **strategie**
*Les stratégies d'exécution des expertises.*
*0 : stratégie par défaut, i.e. saturation (boojum);*
*-1 : dès qu'un déclenchement"valide" est effectué, arrêt de l'exécution de l'expertise;*
*-10 : seules les instanciations mentionnant au moins un fait déduit depuis le dernier examen sont prises en compte;*
*-11 : -1 et -10;*
*-100 : ne sont comptées comme déclenchements valides que ceux modifiant la base de faits ou ayant un effet de bord (écriture, exécution de programme externe ou d'expertise, par exemple);*

*-101 : -1 et -100;*
*-110 : -10 et -100;*
*-111 : -100 et - 11.*

**etape** StrategieDExecution **strategie**
*Stratégie d'exécution d'une étape exécution d'une expertise.*

### 6.2 Exemple

Le plan suivant est très simple : il permet de déterminer si deux ensembles sont égaux.

Nous avons les deux expertises :

```
;-------------------------------------------------------------
-;

expertise EnsemblesNonEgaux


regle EnsemblesDifferents1
tant_que
     Action(Comparer(E) (F) : (comp)) = EnCours
     Element(X) = (E)
     Element(X) = (F) 'inexistant
alors
     Action(comp) <-- Finie
     NonEgal(E) = (F)

regle EnsemblesDifferents2
tant_que
     Action(Comparer(E) (F) : (comp)) = EnCours
     Element(X) = (F)
     Element(X) = (E) 'inexistant
alors
     Action(comp) <-- Finie
     NonEgal(E) = (F)

;-------------------------------------------------------------
-;

expertise EnsemblesEgaux


regle EnsemblesEgaux
tant_que
     Action(Comparer(E) (F) : (comp)) = EnCours
alors
     Action(comp) <-- Finie
     Egal(E) = (F)

;-------------------------------------------------------------
-;
```

Le plan associé à ces deux expertises est

```
(EgaliteDEnsembles PlanPotentielPourBut PlanPourEgaliteDEnsembles)
                        ConditionPlanButContexte nil

(PlanPourEgaliteDEnsembles EtapeDePlan 1) ActionPlan
                        (EnsemblesNonEgaux Executer Expertise)
(PlanPourEgaliteDEnsembles EtapeDePlan 2) ActionPlan
                        (EnsemblesEgaux Executer Expertise)

(PlanPourEgaliteDEnsembles EtapeDePlan 1) ActionInitialePlan
                        PlanPourEgaliteDEnsembles
(EnsemblesEgaux ExpertiseDeclenchee Oui) ConditionButAtteint
                        PlanPourEgaliteDEnsembles

(PlanPourEgaliteDEnsembles EtapeDePlan 1)
    nd-->
(PlanPourEgaliteDEnsembles EtapeDePlan 2)
```

**Remarques :**

- Ce but ne sera considéré comme satisfait que si les deux ensembles sont égaux. Le niveau méta "sait" donc indirectement si les deux ensembles sont égaux. Si ce but sert comme sous-but dans un autre plan, cet autre plan peut ou non en tenir compte.

- L'exécution de ces expertises, pour "fonctionner", doit être précédée de l'installation de faits de la forme

  (**ensemble** Comparer **ensemble**) Action EnCours

  en base de faits initiale, ou plutôt par d'autres expertises. Ces faits définissent de *quels* ensembles étudier l'égalité.

### 6.3 Plan pour exécuter un plan

Ce plan est reproduit dans les trois pages de listing qui précèdent. Les expertises contrôlées par ce plan (les méta-règles d'exécution d'un plan) sont données en annexe.

Ce plan a une particularité : il n'utilise qu'un nombre limité de possibilités du "langage" de buts et plans (pas de sous-buts ni de paquets d'expertises, condition de succès sur le déclenchement d'une expertise uniquement, etc). La raison en est que ce plan est compilé par des expertises en une procédure ensuite incorporée à Shal pour constituer le "contrôle inconscient". Or, ces expertises de compilation n'acceptent qu'un langage de plan limité. Si ces expertises de compilation étaient étendues, elles pourraient compiler n'importe quel plan, et on pourrait donc inclure dans Shal un "contrôle inconscient" pour chaque plan de l'utilisateur (en le faisant agir au niveau méta 1, au lieu de 2 pour le contrôle inconscient général).

Ce plan, dans sa version actuelle, est lié à 29 expertises de Shal comprenant au total 56 règles. La compilation de ce plan en contrôle inconscient est effectuée par 14 expertises regroupant 37 règles, et exécutées par un plan comprenant 11 étapes.

### 6.4 Quelques éléments de contrôle supplémentaires

**action** : **listact**
**action** ListeActions **listact** : **listact**
***listact*** *représente les listes d'actions, c'est-à-dire soit une action unique (premier schéma), soit une liste d'actions (second schéma).*

**listact** Element ActionPonctuelleInterpreteeDeDebutParDefaut
**listact** Element ActionPonctuelleInterpreteeDeFinParDefaut
*Listes d'actions exécutées avant tout début d'inférence au niveau n-2, avec n le niveau de démarrage (niveau du contrôle procédural), ou après la dernière inférence du niveau n-2. Sert par exemple à appeler la fonction qui demande le niveau de trace désiré (début) ou celle imprimant une base de faits (fin).*

**action** ParametreAction **parametre** : **action_parametree**
*Objet représentant une action avec un paramètre (stratégie si expertise, liste de paramètres d'appel si procédure du moteur).*

**action_parametree** Element ActionPermanenteParDefaut
**action** ActionProvisoireAssocieeAAction **action_parametree**
**etape** ActionProvisoireAssocieeAEtape **etape_parametree**
*L'action paramétrée met une certaine variable du moteur a une certaine valeur (ce qui a des effets de bord). Cette valeur est indiquée dans l'ensemble* ActionPermanenteParDefaut. *Une action ou une étape est accompagnée d'un changement provisoire de cette valeur par défaut par les deux derniers schémas de faits qui précèdent. Ceci est utilisé par exemple pour un changement provisoire de niveau de trace.*

**action** ActionDebutantJusteAvantAction **action_parametree**
**etape** ActionDebutantJusteAvantEtape **etape_parametree**
**action** ActionDebutantJusteAprèsAction **action_parametree**
**etape** ActionDebutantJusteAprèsEtape **etape_parametree**
*L'action paramétrée est exécutée juste avant ou après une action ou une étape. Son effet, si elle comporte des effets de bord, reste permanent jusqu'à la prochaine action la défaisant. L'action paramétrée exécutée est en quelque sorte insérée dans les plans comportant l'action ou l'étape mentionnée. On ne peut faire actuellement une utilisation récursive de cette possibilité. Ceci est utilisé par exemple pour un changement de niveau de trace.*

## Références

[Davis 82] Doug Lenat & Randall Davis. *Knowledge-based systems in Artificial Intelligence.* Mc Graw-Hill, 1982.

[Dormoy 87] Jean-Luc Dormoy. *Résolution qualitative : complétude, interprétation physique et contrôle. Mise en œuvre dans un langage à base de règles : Boojum.* Thèse de l'Université Paris 6, décembre 1987.

[Dormoy 89] Jean-Luc Dormoy. *Amélioration de l'efficacité du pattern-matching dans le langage à base de règles Boojum.* Convention IA, janvier 1989. Hermès, éditeur.

[Dormoy 90a] Jean-Luc Dormoy. *Représentation et utilisation des connaissances : contraintes sémantiques sur une base de faits Shal.* Note EDF HI-21/7185, novembre 1990.

[Dormoy 90b] Jean-Luc Dormoy. *Predicting the Behavior of a Knowledge Base.* Cognitiva'90, November 1990, Madrid (Spain). Also to be published by Elsevier.

[Dormoy 91a] Jean-Luc Dormoy. *Connaissances pour compiler des connaissances : le système Shal.* Revue de l'Intelligence Artificielle Vol. 5 n° 4, 1991, Hermès.

[Dormoy 91b] Jean-Luc Dormoy & Sylvie Kornman. *Meta-knowledge, autonomy and (artificial) evolution: some lessons learnt so far.* First European Conference on Artificial Life, ECAL'91, December 1991, Paris (France). Published by MIT Press.

[Laird 87] Laird, Newell & Rosenbloom. *SOAR: an architecture for general intelligence*. Artificial Intelligence 33, 1987, pp. 1-64.

[Laurière 86] Jean-Louis Laurière. *Un langage déclaratif : SNARK.* Technique et Science Informatique, mars 1986.

[Maes 87] *Meta-Level Architectures and Reflection.* Springer-Verlag, 1987. Maes & Nardi, Editors.

[Minton 89] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni & Y. Gil. *Explanation-based learning: a problem-solving perspective*. Artificial Intelligence 40, 1989, pp. 63-118.

[Parchemal 88] Yannick Parchemal. *SEPIAR : un système à base de connaissances qui apprend à utiliser efficacement une expertise.* Thèse de l'Université Paris 6, décembre 1988.

[Pitrat 84] Jacques Pitrat. *MACISTE, un système qui utilise des connaissances pour utiliser des connaissances.* Colloque d'Intelligence Artificielle du Groupe de Recherche C.F. Picard, septembre 1984, Aix en Provence.

[Pitrat 85] Jacques Pitrat. *MACISTE, ou comment utiliser un ordinateur sans utiliser de programme.* Colloque d'Intelligence Artificielle du Groupe de Recherche C.F. Picard, septembre 1985, Toulouse.

[Pitrat 86] Jacques Pitrat. *Le bootstrap sur l'utilisation des connaissances.* Colloque d'Intelligence Artificielle du LAFORIA, septembre 1986, Strasbourg.

[Pitrat 87] Jacques Pitrat. *La déprocéduralisation.* Colloque d'Intelligence Artificielle du LAFORIA, septembre 1987, Cæn.

[Pitrat 88] Jacques Pitrat. *Ceci n'est pas un article.* Colloque Franco-Espagnol d'Intelligence Artificielle de la "Méta-Connaissance", septembre 1988, Areny de Mar (Espagne).

[Pitrat 89] Jacques Pitrat. *Qu'est-ce qu'un individu au royaume de la connaissance?* Colloque d'Intelligence Artificielle sur la Méta-Connaissance, septembre 1989, Le Mans.

[Pitrat 90] Jacques Pitrat. *Métaconnaissance, futur de l'Intelligence Artificielle.* Hermès, 1990.

[Vialatte 85] Michèle Vialatte. *Description et applications du moteur d'inférence SNARK.* Thèse de l'Université Paris 6, mai 1985.