NMS : un module multi-specialistes pour le generateur de systemes experts Smeci

Olivier Corby

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Rapports Techniques

N°146

Programme 2

Calcul symbolique, Programmation et Génie logiciel

NMS : UN MODULE MULTI-SPÉCIALISTES POUR LE GÉNÉRATEUR DE SYSTÈMES EXPERTS SMECI

Olivier CORBY

Octobre 1992

NMS : Un module multi-spécialistes pour le générateur de systèmes experts Smeci

NMS : a multi-specialist module for the Smeci expert system shell

Olivier CORBY

Résumé

Ce rapport est le manuel de référence du logiciel NMS qui est un module exécutable dans le générateur de système expert Smeci. NMS apporte à Smeci le concept de *spécialiste* qui permet de structurer et de modulariser une base de connaissances, dans le cadre général des architectures de type blackboard.

Abstract

This report is the reference manual of the NMS software module which runs within the Smeci expert system shell. NMS introduces the *specialist* concept which enables the structuring and modularization of a knowledge base, within the blackboard framework.

Chapitre 1

Introduction

Le générateur de systèmes experts Smeci [II91] offre une représentation des connaissances sous forme d'objets, de règles et de tâches. Les bases de connaissances Smeci peuvent ainsi être modularisées autour du concept de *tâche*. Mais ce formalisme présente quelques limitations. Il prévoit une décomposition essentiellement descendante des problèmes et un enchaînement défini statiquement. De plus, il n'y a qu'un seul point d'entrée qui est la racine de l'arbre des tâches.

Enfin, l'enchainement est décrit explicitement. Il n'y a pas de déclaration de compétences des tâches. On aimerait en fait pouvoir spécifier des interactions déclaratives, basées sur la sémantique des tâches.

Un programme Smeci, appelé NMS, permettant de modulariser une base de connaissances en modules appelés *spécialistes*, a été développé. Les spécialistes déclarent leurs compétences et interagissent par l'intermédiaire de requêtes.

NMS (Noyau Multi-Spécialistes), a été initialement conçu pour le système expert Erasmus, en collaboration avec le CETE Méditerranée, dans le cadre d'un contrat de recherche financé par le SETRA[1]

1.1 Motivations

Les motivations pour la conception de NMS sont les suivantes.

[1]Convention No 1 91 E242 entre le Service d'Etudes Techniques des Routes et Autoroutes et l'INRIA.

Tout d'abord NMS permet de modulariser la base de connaissances. Erasmus incorpore la connaissance d'une vingtaine d'experts, sous forme de spécialistes.

L'architecture permet une mise au point incrémentale du système. Il est aisé d'ajouter de nouveaux spécialistes.

Enfin, le domaine d'application d'Erasmus ne permet pas de définir un seul arbre de tâches global. Au contraire, le calcul de l'enchaînement des tâches est dynamique.

Cette extension à Smeci comporte trois modules compatibles.

NMS *minimal* définit un protocole pour implanter des spécialistes avec des déclarations de compétences, communiquant par envoi de requêtes. Les concepts principaux de NMS tels que les spécialistes y sont définis.

NMS *non monotone* permet la reprise automatique de raisonnements sur changement d'hypothèses.

NMS *étendu* implante le langage de requête initialement développé pour Erasmus comprenant des requêtes spécialisées permettant de déterminer ou de suspecter un champ d'un objet.

Le logiciel NMS est conçu de manière à pouvoir être étendu.

1.2 Principales fonctions de NMS

Un spécialiste fédère des connaissances permettant de résoudre un ensemble de sous-problèmes voisins.

Le mode d'interaction des spécialistes est la *coopération volontaire*. Les différents spécialistes interagissent de manière non conflictuelle, ont le même objectif (résoudre le problème) et partagent l'information sans discrimination.

Plus précisément, en utilisant la classification de Hautin, [HV86], la coopération se fait suivant le modèle de l'*appel d'offre*, avec ou sans compétition. Un spécialiste peut sous-traiter un travail à un ou plusieurs autres spécialistes.

Afin de conserver la modularité du système, chaque spécialiste ne connaît pas les compétences respectives des autres spécialistes. Par contre, un spécialiste peut demander, selon un protocole standard, la résolution d'un sous-problème. Un, ou plusieurs, spécialistes répondent à la demande et résolvent le problème sur une base d'objets

commune. Le spécialiste appelant a ainsi accès au résultat obtenu.

Nous proposons le partitionnement de la connaissances autour du concept de spécialiste composé de méthodes, de règles d'objets et de tâches. Les spécialistes opèrent sur une base de données commune. Les spécialistes utilisent ainsi les mêmes formalismes de représentation des connaissances.

Le système doit pouvoir fonctionner même si certains modules sont absents, pas encore développés ou non encore intégrés. Si aucun spécialiste ne peut traiter un sous-problème, le système doit pouvoir s'adresser à l'utilisateur, via une interface redéfinissable.

L'arbitrage des conflits doit pouvoir être redéfini pour traiter le cas où plusieurs spécialistes pourraient traiter un même sous-problème. L'arbitrage doit pouvoir être programmé à base de connaissances, sous forme de règles expertes par exemple, ou bien générer des raisonnements alternatifs en parallèle.

Enfin, le raisonnement doit pouvoir suivre un mode de contrôle hiérarchique à la Crysalis [Ter83], car les sous-problèmes à traiter peuvent être décomposés de manière descendante. Un spécialiste pourra traiter un problème au moyen d'un arbre de tâches.

L'architecture de NMS s'inspire d'un prototype développé initialement par le CETE Méditerranée et a fait l'objet de publications conjointes [CAN90, FC88].

Les interactions entre spécialistes sont déclaratives, par opposition à des appels impératifs à telle procédure de tel spécialiste nommés. Ici, chaque spécialiste déclare ses propres compétences et un spécialiste spécifique, le superviseur, achemine la requête de résolution de problème aux spécialistes compétents.

Un spécialiste émettant une requête est interrompu le temps du traitement de la requête et est restauré ensuite. Le traitement des requêtes est donc séquentiel.

La réponse à une demande de résolution de problème est une tâche, ou un ensemble de tâches.

1.3 Applications

NMS est maintenant utilisé, outre par Erasmus, dans deux autres applications. La première est un système d'interprétation de scènes

dans un contexte de fusion multi-capteurs appliqué au traitement d'images de télédétection, effectué au sein du projet Pastis de l'INRIA [CGHS92].

La deuxième application concerne la formalisation de connaissances pour un système de détermination et de validation de résultats d'expériences de laboratoire sur des études rhéométriques sur les polymères [Neg92]. Ce travail est effectué au CEMEF de l'Ecole des Mines de Paris à Sophia-Antipolis.

Chapitre 2

NMS minimal

2.1 Introduction

NMS minimal définit les concepts de `Specialiste`, `Competence` et `Superviseur` ainsi que le protocole d'extension du noyau.

Du point de vue de l'utilisateur, le superviseur sert à déclarer l'ensemble des spécialistes actifs dans l'application. Un spécialiste est une collection de compétences qui sont elles mêmes implantées sous forme de tâches Smeci.

NMS minimal définit un protocole pour étendre la définition des compétences avec une partie *déclaration de compétence*. Pour cela, il est nécessaire de définir une sous classe de `Competence` avec des champs spécifiques pour la déclaration. Il faut également associer une méthode `traite` (cf p. 12) qui teste si une compétence peut traiter une requête.

L'émission d'une requête se fait par la primitive `nms-requete` qui comporte une liste d'arguments nommés :

```
(nms-requete 'probleme SP12
    'objet Digue-23 'type 'optimisation)
```

Les arguments des requêtes dépendent de l'application et du langage de déclaration de compétence choisi. NMS n'impose aucun argument.

2.1.1 Traitement des requêtes

Le traitement d'une requête se fait en deux étapes.

La première permet de sélectionner la compétence susceptible de traiter la requête. La deuxième consiste à interpréter les tâches associées à l'aide du moteur d'inférence de Smeci.

Sélection

La première étape a lieu dans l'état Smeci où la requête est émise. On appelle *émission* d'une requête le faite d'exécuter une des primitives de type `nms-requete`.

NMS consulte chaque compétence et teste si elles sont susceptibles de traiter la requête. Le test se fait au moyen de la méthode `traite` qui prend en arguments la compétence et la liste des arguments d'appels de la requête.

Si un raisonnement a déjà été effectué par une compétence avec les mêmes arguments que la requête courante, cette compétence n'est pas sélectionnée (sauf si cela est explicitement demandé par l'argument `nms-reprise`, cf p. 14).

Si plusieurs compétences sont candidates, NMS en choisit une au moyen d'une fonction de choix qui lui a été fournie. La fonction de choix prend en premier argument la liste des compétences candidates et en deuxième argument la liste des arguments de la requête.

La fonction de choix renvoie en résultat une compétence. L'élimination des compétences déjà traitées est faite avant l'appel à la fonction de choix.

Il existe une fonction de choix par défaut qui choisit une compétence "au hasard", en fait la première de la liste. On peut préciser une fonction de choix pour la requête courante au moyen de l'argument système de nom `nms-choix` (cf p. 13). L'effet est le suivant : si plusieurs compétences sont candidates pour traiter la requête émise, on en choisit une au moyen de la fonction de choix qui est indiquée en argument.

Il est également possible d'indiquer une fonction de choix de manière globale par la primitive `nms-choix` (cf p. 13). Dans ce cas, la fonction de choix utilisée par défaut est celle indiquée par la primitive `nms-choix`.

Traitement

La deuxième phase, appelée traitement, consiste à transmettre au moteur Smeci les tâches de la compétence choisie, en vue de les interpréter. Elle commence par l'interruption du raisonnement courant et la sauvegarde des tâches Smeci correspondantes qui seront restaurées à la fin de l'interprétation des tâches de la compétence.

Cette phase a lieu *dans le cycle de raisonnement suivant* le cycle durant lequel la requête a été émise. Le traitement d'une requête n'est donc pas fonctionnel mais produit des effets de bords dans la base d'objets. On ne peut donc pas émettre de requête dans une prémisse de règle.

Ainsi, une règle qui émet une requête ne peut obtenir directement les résultats de la requête. Ceux ci ne seront disponibles qu'à la suite d'un raisonnement. Une autre règle doit donc récupérer ces résultats. Une même rêgle peut par contre émettre plusieurs requêtes qui seront traitées dans l'ordre de leur émission.

N.B. : une règle qui émet une requête ne doit pas modifier ensuite l'agenda des tâches. En effet, l'émission d'une requête a pour effet d'empiler une tâche système (i.e. le superviseur) qui prendra en charge le traitement, dans le cycle suivant.

2.2 Interface fonctionnelle

`(nms-requete` {`nom valeur`}$^+$`)`

Pour émettre une requête de manière déclarative. `nom` est le nom d'un argument à transmettre et `valeur` est sa valeur. Cette primitive lance le moteur Smeci si celui-ci est à l'arrêt.

Les compétences sont sélectionnées au moyen de la méthode `traite` (cf p. 12).

`(nms-requete-cible cible` {`nom valeur`}*`)`

cible : nom ou objet {specialiste | competence}

Pour émettre une requête de manière impérative, en précisant le spécialiste ou la compétence destinataire.

On s'assure également que le destinataire peut traiter la requête, au moyen de la méthode traite.

(nms-requete-p {nom valeur}$^+$)

Cette primitive permet de tester si une requête est susceptible d'être traitée. Elle renvoie une liste de compétences.

(nms-valeur nom [larg | Mission])

Fonction à un ou deux arguments qui permet de récupérer la valeur d'un argument de nom `nom` passé lors de l'émission d'une requête.

Avec un seul argument, récupère la valeur de l'argument `nom` du raisonnement courant (de la requête en cours de traitement). On peut donc référencer les arguments de la requête dans les règles d'inférences du spécialiste retenu.

Dans certains cas, on veut récupérer la valeur d'un argument à partir d'une liste d'arguments, comme dans la méthode `traite` ou dans la fonction de choix. On peut passer cette liste en deuxième argument de la fonction `nms-valeur`.

On peut également récupérer un argument d'une des requêtes déjà traitées. Cela se fait par l'intermédiaire des missions (cf p. 13). Un appel à `nms-valeur` avec une mission en deuxième argument récupère l'argument de nom `nom` de la requête correspondante.

Dans tous les cas, la primitive déclenche une erreur si l'argument de nom `nom` est indéfini.

(nms-valeur-p nom [larg | Mission])

Permet de tester si un argument est défini.

(nms-superviser)

Invoque le superviseur de NMS (cf p. 20).

2.3 Catégories système

Cette section présente les principales classes de NMS .

2.3.1 Superviseur

Le superviseur possède trois champs utilisateur.

Champs

`specialistes` : `lobject Specialiste.nms`

Contient la liste des spécialistes connus du superviseur.

`generaliste` : `object Specialiste.nms`

Contient le spécialiste généraliste utilisé dans la méthode `mediter` définie ci-dessous.

`mission-generaliste` : `symbol`

Contient le nom de la compétence du généraliste à étudier en cas de blocage du superviseur (cf p. 11).

Méthodes

`(send 'mediter superviseur)`

Le superviseur reçoit ce message lorsqu'il n'y a plus de requête à traiter. Par défaut, ce message lance le spécialiste *généraliste*, qui se trouve dans le champ `generaliste` du superviseur, avec le nom de la compétence qui se trouve dans son champ `mission-generaliste` .

Cette méthode est redéfinissable.

2.3.2 Specialiste

Un spécialiste contient un champ utilisateur :

`guichets : lobject Competence.nms`

contient la liste des compétences.

2.3.3 Competence

Une compétence est un objet comportant un champ et une méthode utilisateur.

Champ

`reponse : lobject S.Task`

contient la liste des tâches qui seront invoquées en réponse à la requête.

Méthode

`(send 'traite competence larg)`

`larg : aliste des arguments de la requête.`

Doit renvoyer `t` si la compétence accepte la requête et `()` sinon. Cette méthode doit être définie pour toute nouvelle classe de compétence définie par l'utilisateur.

La aliste des arguments peut être consultée par la primitive :

`(nms-valeur nom larg)`.

2.4 Les arguments système

Cette section décrit les arguments système prédifinis qui existent dans NMS.

nms-choix

Cet argument permet de préciser une fonction de choix des compétences en réponse à une requête.

La fonction de choix prend deux arguments `lcomp` et `larg`, où `lcomp` est la liste des compétences candidates et `larg` est la liste d'arguments de la requête dans un format consultable par la fonction `nms-valeur`. La fonction de choix doit renvoyer une des compétences en résultat.

(nms-choix [fonction])

Cette fonction permet de spécifier la fonction de choix par défaut.

nms-discrimine

Une compétence est invoquée au moyen d'un objet *mission* qui contient, entre autre, la liste des arguments de la requête et la compétence choisie. Deux occurrences successives de la même requête traitées par la même compétence sont invoquées dans la même mission alors que deux requêtes différant par au moins un argument sont traitées dans des missions différentes.

Toutefois, on peut estimer que deux requêtes successives dont un sous-ensemble des arguments, appelés *discriminants*, est identique doivent être invoquées dans la même mission (pour des questions d'économie par exemple). Dans ce cas, on perd la trace de la première requête.

L'argument `nms-discrimine` permet de spécifier les arguments discriminants d'une requête.

Par exemple, les deux requêtes suivantes sont invoquées dans des missions différentes :

```
(nms-requete 'probleme 'p12 'delta 10)
```

```
(nms-requete 'probleme 'p12 'delta 20)
```

Alors que les deux requêtes suivantes sont invoquées dans la même mission :

```
(nms-requete 'probleme 'p12 'delta 10)
(nms-requete 'probleme 'p12 'delta 20
    'nms-discrimine '(probleme))
```

(nms-discrimine liste-argument)

La fonction `nms-discrimine` permet de spécifier les arguments discriminants par défaut. Si on précise une valeur pour l'argument `nms-discrimine` dans une requête, cette valeur masque la valeur par défaut. Si rien n'est précisé, tous les arguments sont discriminants.

nms-reprise

Cet argument permet de moduler le comportement de NMS au cas où une requête ayant déjà fait l'objet d'un traitement est réémise. Il est possible de préciser si l'on veut reprendre une telle requête ou si l'on ne souhaite pas la traiter à nouveau.

L'argument a deux valeurs possibles :

- `()` : on ne la reprend pas si elle est déjà traitée (mode par défaut),
- `t` : on la reprend entièrement.

Par compatibilité avec les versions précédentes, on conserve la valeur suivante de l'argument `nms-reprise` qui doit être utilisée avec modération :

- `p` : on la reprend là où elle est si elle est interrompue, et entièrement si elle est terminée.

(nms-reprise [() | p | t])

Cette fonction permet de préciser le mode de reprise par défaut.

2.5 Les erreurs

- `errnms-notasupervisor` Le superviseur de NMS est invalide, recharger NMS.
- `errnms-notaspecialist` La cible de la requête n'est pas un spécialiste connu.
- `errnms-notalabellist` La liste d'arguments de la requête est invalide (nombre impaire d'arguments).
- `errnms-notalabel` L'argument n'est pas connu (nms-valeur).

2.6 Etendre NMS minimal

On peut étendre la définition des compétences en associant des informations telles que le *problème*, le *contexte*, la *fiabilité*, etc. On tient compte de ses informations pour décider si une compétence peut traiter une requête.

```
(defScategory Savoir Competence.nms
    (probleme
        allocation constrained
        type enum-sumbol
        range (p-1 p-2 .. p-n))
    (contexte
        allocation constrained
        type enum-sumbol
        range (facile moyen difficile))
    (fiabilite
        allocation constrained
        type enum-symbol
        range (faible moyenne forte)))
```

On redéfinit la méthode `traite` qui estime si une compétence peut traiter une requête en fonction de la valeur de certains arguments, ici `probleme` et `contexte`.

```
(defSmethod {Savoir}:traite(s larg)()
    (and (eq (nms-valeur 'probleme larg) probleme^s)
         (memq (nms-valeur 'contexte larg) contexte-l^s)   ...))
```

On peut imaginer également que chaque compétence possède un prédicat permettant de tester sa capacité à traiter la requête. Dans ce cas la compétence possède un champ `predicat` et la méthode `traite` appelle ce prédicat.

Chapitre 3

NMS non monotone

3.1 Introduction

Les raisonnements des spécialistes peuvent dépendre de certaines hypothèses. Si une hypothèse est remise en cause, les raisonnements qui en dépendent doivent être recommencés en prenant en compte la nouvelle hypothèse.

Au sein de NMS, *une hypothèse est la valeur d'un champ d'un objet*. La remise en cause d'une hypothèse est implantée sous forme du changement de la valeur d'un champ d'un objet.

Les spécialistes peuvent déclarer les hypothèses desquelles dépend la validité de leurs raisonnements. Ceci peut être fait statiquement et faire partie de la connaissance du système expert : la validité du diagnostic dépend de l'*intensité* du *trafic*.

La déclaration d'hypothèses peut également être faite en cours de raisonnement. En effet, certains objets dont les champs sont des hypothèses peuvent ne pas exister dans l'état de départ du raisonnement et être créés par déduction.

3.2 Traitement

Lorsqu'un champ d'un objet hypothétique change de valeur, NMS calcule l'ensemble des raisonnements qui en dépendent et les enregistrent comme étant *à refaire*. Le calcul des raisonnements à refaire s'effectue sur la base des missions déjà réalisées (cf p. 13).

Par défaut, les raisonnements sont refaits lors du prochain passage dans le superviseur (à l'occasion du traitement d'une requête ou bien lorsque plus aucune requête n'est à traiter).

Cependant, il est possible de reprendre les raisonnements concernés par un changement d'hypothèse dans le cycle suivant le changement (cf la méthode `changement-hypothese` p. 20). Dans ce cas, le raisonnement courant est interrompu.

Suite à un changement d'hypothèse, un raisonnement est recommencé avec la base d'objets en l'état. C'est à dire qu'il n'y a pas de retour arrière ou de remise en cause de certaines déductions. On relance simplement les tâches ayant réalisé le raisonnement précédent.

Lorsque plusieurs raisonnements doivent être refaits, on commence par les raisonnements les plus anciens.

Si un raisonnement émet une requête qui a déjà été traitée et qui est elle aussi à refaire pour cause de changement d'hypothèse, on ne retraite cette requête qu'une seule fois.

NMS non monotone est une extension de NMS minimal. Il introduit les classes `HCompetence.nms`, `HSuperviseur.nms` et `Hypothese.nms`.

3.3 Catégories système

3.3.1 Hypothese

Les hypothèses se déclarent sous forme d'instance de la classe `Hypothese.nms` qui possède les champs suivants :

`objet` : `object S.Object`

contient l'objet sur lequel porte l'hypothèse.

`champ` : `symbol`

contient le nom du champ hypothétique.

`valeur` : `method`

calcule la valeur courante de l'hypothèse.

3.3.2 HCompetence

Extension de `Competence.nms` permettant de déclarer la dépendance par rapport à des hypothèses. Cette classe possède le champ supplémentaire suivant :

hypotheses : `lobject Hypothese.nms`

Les raisonnements effectués par des `HCompetence` sont automatiquement rendus sensibles aux hypothèses déclarées dans le champ `hypothèses`.

3.4 Interface fonctionnelle

(nms-hypothese-cible `hcom objet champ`)

Permet de spécifier que la validité des raisonnements futurs de la compétence `hcom` dépend de la valeur de `champ` de `objet`. `hcom` doit être de type `HCompetence.nms`.

(nms-hypothese `objet champ`)

Permet de spécifier que la validité du raisonnement actuel du spécialiste courant dépend de la valeur de `champ` de `objet`.

```
(send 'reprise-raisonnement-p superviseur
      objet champ aval nval)
```

Méthode redéfinissable, invoquée avant le calcul des raisonnements à reprendre suite au changement de l'hypothèse `champ` de `objet`. `aval` est l'ancienne valeur et `nval` est la nouvelle valeur du champ de l'objet modifié.

On ne reprend ces raisonnements que si la méthode renvoie autre chose que `()`. Par défaut, elle renvoie `t`, et on reprend donc tous les

raisonnements.

```
(send 'changement-hypothese superviseur
      objet champ aval nval)
```

Méthode redéfinissable, invoquée après le calcul des raisonnements à reprendre suite au changement de l'hypothèse `champ` de `objet`. Cette méthode pourrait invoquer le superviseur par `(nms-superviser)` pour traiter les raisonnements à refaire dans le cycle suivant.

3.5 Erreurs

- `errnms-notanobject` : L'objet n'est pas un objet Smeci, dans la déclaration d'une hypothèse.
- `errnms-notacompetence` : L'argument de la déclaration d'hypothèse n'est pas une compétence connue.

Chapitre 4

NMS étendu

4.1 Introduction

Ce module correspond à celui qui a été développé dans le cadre du projet Erasmus. Il implante une extension du modèle des compétences et des requêtes.

La déclaration de compétence se fait de la manière suivante : un spécialiste peut déclarer être capable de déterminer un champ des instances d'une catégorie particulière. Par exemple : déterminer le *modèle* du *béton bitumineux*.

Notons qu'il est possible de préciser des catégories abstraites comme *Probleme*, pour lesquelles on aura défini une hiérarchie de classes de problèmes et des attributs comme *solution*, *vraisemblance*, *coût*, etc. Un spécialiste déclare alors pouvoir déterminer la solution d'une certaine classe de problème.

Les compétences s'expriment sous forme de *guichets*, qui sont des instances de la classe `Guichet.nms`. Les guichets sont des compétences sensibles à des hypothèses, en conséquence, NMS étendu repose sur NMS non monotone. Enfin, les requêtes s'expriment grace à une extension de l'interface fonctionnelle.

4.2 Les classes

4.2.1 Guichet

Un guichet est une spécialisation de `Competence.nms` comportant deux champs utilisateur.

Les guichets d'Erasmus étant sensibles à des hypothèses, la catégorie `Guichet.nms` est une sous catégorie de `HCompetence.nms`.

Voci les champs des guichets :

`classe` : `symbol`

contient un nom de catégorie. Le spécialiste est susceptible de déterminer les valeurs de certains champs des instances (directes ou indirectes) de cette catégorie.

`champ_l` : `lsymbol`

contient une liste de champs que le spécialiste est susceptible de déterminer, pour les instances de la catégorie ci-dessus. Il est possible de ne pas préciser de valeur pour ce champ.

4.2.2 GuichetSuspicion

Cette classe permet de déclarer la connaissance qui traite les suspicions (cf p. 24). C'est une spécialisation de `Guichet.nms`.

4.3 Interface fonctionnelle

4.3.1 Emission d'une requête

Cette section décrit les primitives disponibles pour émettre des requêtes. Ce sont des fonctions lisp que l'on peut appeler dans les conclusions des règles, dans des méthodes ou bien sous l'interprète Le-Lisp.

Les primitives `determiner` et `suspecter` ne renvoient pas en résultat la valeur du champ. En effet, elles ont pour effet de lancer un raisonnement. De plus, elle n'ont pas pour conséquence d'affecter le

champ de l'objet avec une valeur. C'est au raisonnement de le faire éventuellement.

Ces primitives sont implantées sous forme d'appels aux primitives de NMS minimal telles que `nms-requete`, en positionnant des arguments systèmes prédéfinis.

`(determiner objet champ . larg)`

`objet` est un objet Smeci, `champ` est un nom de champ de `objet` et `larg` est une liste optionnelle d'arguments formés de couples `nom valeur`. Cette primitive permet de lancer un spécialiste susceptible de déterminer la valeur du champ de l'objet. Il est possible de ne pas préciser de champ, auquel cas on cherchera un guichet approprié n'ayant pas précisé de champ.

`(determiner-spe nom-spe objet champ . larg)`

`nom-spe` est le nom d'un spécialiste, `objet` est un objet Smeci et `champ` est le nom d'un champ de l'objet.

La fonction a le même comportement que la précédente, tout en précisant le spécialiste destinataire. Seul ce dernier recevra la requête.

`(objet-courant)`

L'objet passé en argument des fonctions `determiner` peut être récupéré, lors du raisonnement correspondant au traitement de la requête, par un appel à cette fonction. On pourra par exemple écrire :

```
defregle r18
soit o un Objet parmi $(objet-courant)
si longueur^o <> ()
...
fin-regle
```

`(determine objet champ . larg)`

Renvoie une liste d'instances de la classe `Guichet.nms` capables

de déterminer la valeur du champ de l'objet.

(etudier `nom-guichet . larg`**)**

`nom-guichet` est un nom de guichet.

Lance l'exécution des tâches du guichet, sans qu'il soit besoin de préciser un objet ou un champ. Dans ce cas, l'appel à la fonction `(objet-courant)` renvoie `()`.

(etudier-spe `nom-spe nom-guichet . larg`**)**

`nom-spe` est le nom d'un spécialiste, `nom-guichet` est un nom de guichet.

Autres fonctions

Les versions suivantes sont gardées par compatibilité. Elles correspondent à des appels aux primitives précédentes avec l'argument système `nms-reprise-p`.

(determiner+ `objet champ . larg`**)**
(determiner-spe+ `nom-spe objet champ . larg`**)**
(determine+ `objet champ . larg`**)**
(etudier+ `nom-guichet . larg`**)**
(etudier-spe+ `nom-spe nom-guichet . larg`**)**

Par compatibilité ascendante, on peut passer en argument un objet spécialiste au lieu du nom d'un spécialiste dans les fonctions de type `determiner-spe`.

4.3.2 Emission d'une suspicion

Une suspicion est une requête spécialisée à trois arguments, traitée par des compétences spécifiques de type `GuichetSuspicion.nms`. Les suspicions n'ont pas de sémantique particulière dans NMS par rapport aux autres requêtes.

Les suspicions ont été introduites pour le système Erasmus. Elles permettent de *suspecter* la valeur d'un champ d'un objet d'avoir

une certaine propriété, comme d'être sous-évaluée par exemple. Ceci entraîne certains raisonnements spécifiques de l'application.

Les fonctions suivantes permettent d'émettre des suspicions. Elles n'affectent pas le champ de l'objet suspecté et se contentent de lancer un raisonnement si un guichet convient.

La valeur de la suspicion peut être récupérée par la fonction `suspicion?` détaillée plus loin.

Voici l'interface fonctionnelle des suspicions :

`(suspecter objet champ valeur)`

`objet : S.Object, champ : symbol, valeur : *`

Permet de suspecter champ de objet, sans préciser de spécialiste. Ce dernier est déterminé par le superviseur, en fonction des compétences déclarées dans les guichets. Les arguments `objet` et `champ` sont discriminant mais pas l'argument `valeur`. Ainsi, deux suspicions qui ne diffèrent que par la valeur sont traitées par la même mission.

`(suspecte objet champ valeur)`

Renvoie une liste d'instances de `GuichetSuspicion.nms` qui indique les guichets susceptibles de traiter la suspicion.

`(suspecter-spe nom-spe objet champ valeur)`

`nom-spe : nom de Specialiste.nms, objet : S.Object,`
`champ : symbol, valeur : *`

Permet de suspecter `champ` de `objet` en précisant le spécialiste destinataire.

`(suspicion? objet champ [valeur])`

`objet : S.Object, champ : symbol, valeur : *`

Permet de tester si l'on est en train de traiter une suspicion sur `champ` de `objet`, dans la mission courante ou dans une des missions interrompues. Si on ne précise pas de valeur, `suspicion?` renvoie

la valeur de la suspicion, sinon on teste si la valeur de la suspicion courante vaut `valeur`.

Les versions suivantes correspondent à des appels aux primitives précédentes avec l'argument système `nms-reprise p`.

(suspecter+ `objet champ valeur`**)**
(suspecte+ `objet champ valeur`**)**
(suspecter-spe+ `nom-spe objet champ valeur`**)**

Références

[CAN90] O. Corby, F. Allez, and B. Neveu. A multi-expert system for pavement diagnosis and rehabilitation. *Transportation Research International Journal, Special Issue on Expert Systems, S. Ritchie Ed.*, 24A(1), 1990.

[CGHS92] V. Clément, G. Giraudon, S. Houzelle, and F. Sandakly. Interpretation of remotely sensed images in a context of multisensor fusion using a multi-specialist architecture. Rapport de recherche RR-1768, INRIA, 1992.

[FC88] M. Fajon and O. Corby. Erasme, entretien routier assisté par système multi-expert. In *8e Journées internationales sur les systèmes experts*, Avignon, juin 1988. EC2.

[HV86] F. Hautin and A. Vailly. La coopération entre systèmes experts. In *Journées nationales sur l'intelligence artificielle*, pages 187–204. PRC GRECO IA, 1986.

[Ilo91] Ilog and INRIA. *Smeci 1.54 : Le manuel de référence.* Gentilly, 1991.

[Neg92] V. Negru. Modélisation des connaissances en vue de la détermination des paramètres rhéologiques de matériaux polymères. Rapport de DEA, Université de Nice-Sophia Antipolis, Ecole nationale supérieure des Mines de Paris, septembre 1992.

[Ter83] A. Terry. The CRYSALIS Project : Hierarchical Control of Production Systems. Tech. Rep. HPP-83-19, Heuristic Programing Project, Stanford University, 1983.

Table des matières