

---

# Configuration de middleware dirigée par les applications

Vivien Quema<sup>1</sup> — Luc Bellissard<sup>2</sup>

<sup>1</sup>INRIA Rhône-Alpes ; Sardes Project

<sup>2</sup>ScalAgent Distributed Technologies

655, Avenue de l'Europe

F-38334 Saint-Ismier Cedex, France

Tel : (33) 476 615 268 - Fax : (33) 476 615 252

{Vivien.Quema, Luc.Bellissard}@inrialpes.fr

---

*RÉSUMÉ. L'émergence de nouveaux équipements aux ressources limitées complexifie la construction des applications réparties. Les middlewares facilitent le travail des développeurs en prenant en charge les contraintes de la distribution et en offrant un nombre croissant de services. En contrepartie, leur configuration devient une tâche de plus en plus complexe, souvent laissée à la charge du développeur. Nous proposons d'automatiser ce processus de configuration du middleware en utilisant le pouvoir de configuration du middleware orienté messages (MOM) ScalAgent. L'algorithme proposé utilise des informations issues d'une description de l'application dans un langage de description d'architecture (ADL). Il permet de choisir les composants du middleware à installer sur chaque site et la localisation des composants applicatifs. Il vise, en outre, à minimiser la surcharge induite par le middleware à l'exécution. Le gain obtenu grâce à cette configuration dirigée par l'application est évalué par une série de tests de performances.*

*ABSTRACT. There is a growing trend in developing applications on distributed heterogeneous devices. Middleware technology provides many solutions allowing to hide the management of the distribution of services and computations to the developers. On the other hand the middleware configuration becomes more and more complex. We propose to ease the configuration process using the configuration capability of a component-based message-oriented middleware (the Scalagent MOM). Our solution uses the application description (into an extended ADL) to determine the set of middleware components required and their attributes to ensure non-functional properties required by the application. The customization process is controlled by a customization algorithm that tries to minimize some non-functional property management costs. Our performance measurements clearly show the customization advantages.*

*MOTS-CLÉS : middleware à composants, configuration, langage de description d'architecture*

*KEYWORDS: component-based middleware, configuration, architecture description language*

---

## 1. Introduction

Une nouvelle génération de matériel intelligent et communiquant émerge : les téléphones mobiles, les assistants digitaux personnels en sont des exemples. Ces équipements possèdent une puissance de calcul conséquente et sont connectés à l'Internet avec diverses technologies réseaux. Cette évolution technologique permet de développer des applications faisant intervenir ces nouveaux équipements. Les défis techniques soulevés par ces environnements émergents sont l'hétérogénéité, la scalabilité, l'ouverture vers les systèmes informatiques existants. Il est, de plus, souhaitable d'utiliser les mêmes composants applicatifs sur l'ensemble du système afin d'éviter de multiplier les développements de composants et de les spécialiser selon la plate-forme d'exécution. L'idée actuelle symbolisée par le langage Java et les déclinaisons de machines virtuelles (J2SE, J2ME et profils) est de fournir une infrastructure logicielle middleware qui s'adapte aux plates-formes d'exécution et qui permet de conserver les mêmes composants applicatifs.

De nos jours, l'utilisation de communications asynchrones par le biais d'un middleware orienté messages est reconnu comme un des seuls moyens de garantir les spécificités sus-citées (scalabilité, extensibilité, etc.). Il est aussi communément admis que la facilité de configuration des middlewares s'obtient grâce à l'utilisation de techniques de programmation à base de composants. Nous décrivons dans cet article l'utilisation faite, dans l'infrastructure ScalAgent, de ces deux technologies pour permettre une configuration de haut niveau des applications impliquant des équipements divers.

Notre travail consiste à proposer une configuration automatique de l'infrastructure middleware ScalAgent en fonction des besoins applicatifs. Dans l'implémentation actuelle de la plate-forme, le concepteur de l'application distribuée dispose d'un langage de description d'architecture (ADL) [ISS 98b] lui permettant de définir la structure de l'application. Cette description de la structure de l'application est ensuite manipulée par le service de déploiement qui déploie et active les composants applicatifs sur les différents sites. Cependant, l'implémentation de la plate-forme d'exécution ScalAgent, pour une application donnée, est figée et homogène. Cela signifie que les propriétés telles que la persistance des composants applicatifs, l'ordonnement des messages échangés, etc. sont prédéfinies et imposées à chaque nœud du système. Or, du fait de la limitation des ressources de certains équipements, il n'est pas souhaitable, ni même parfois possible, que le middleware responsable de l'exécution et des communications des composants applicatifs fournisse à tous les composants, de façon systématique, le support pour toutes les propriétés non fonctionnelles.

Notre objectif est de lever cette limitation en générant un middleware adapté aux ressources disponibles sur les différents nœuds, et aux exigences des composants applicatifs. Pour atteindre cet objectif, des modifications ont été faites au middleware ScalAgent [BEL 99] et à l'ADL utilisé pour décrire l'application [BEL 00]. Ce dernier a été étendu de façon à permettre au développeur de l'application de spécifier les propriétés non fonctionnelles requises par l'application. D'autre part, le middleware ScalAgent a été ré-architecturé pour faciliter sa configuration (i.e. la possibilité de

changer des fonctions internes du middleware). Enfin, un système de configuration a été conçu pour déterminer la configuration du middleware ScalAgent la mieux adaptée aux besoins des applications spécifiés dans l'ADL.

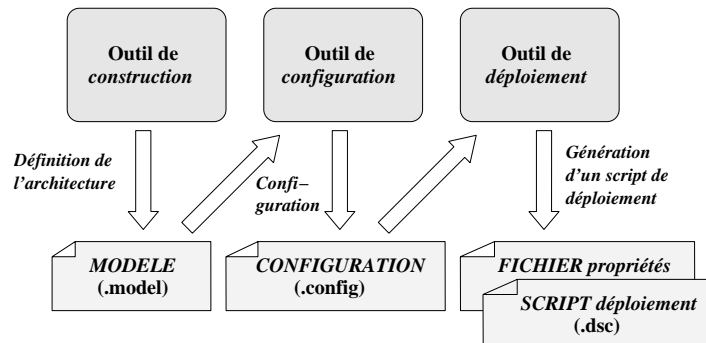
L'article est structuré de la façon suivante : la section 2 est consacrée à la présentation de l'infrastructure ScalAgent ; dans la section 3, nous présentons les extensions faites au langage de description d'architecture ; la section 4 est dédiée à l'algorithme de configuration du middleware et nous présentons quelques tests de performances dans la section 5 ; la section 6 conclut cet article.

## 2. Extension de l'infrastructure ScalAgent

### 2.1. Présentation

L'infrastructure ScalAgent est un support de construction, de configuration et de déploiement d'applications réparties à base de composants. Elle est composée des éléments suivants :

- Le middleware orienté messages (MOM) ScalAgent qui fournit un service de transfert de messages asynchrone ;
- Un ensemble d'outils graphiques reprenant les concepts des langages de description d'architecture.



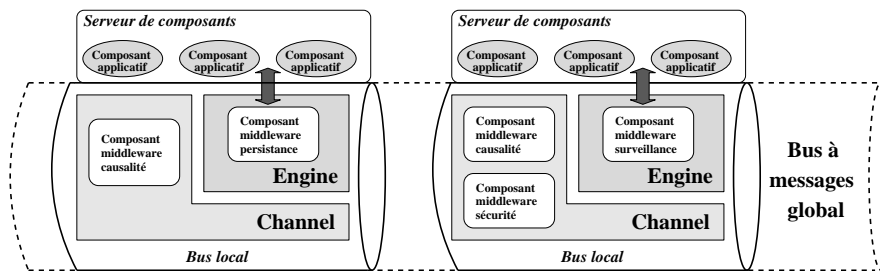
**Figure 1.** Chaîne de construction d'une application avec l'infrastructure ScalAgent

Une application est construite en trois phases (c.f. figure 1). Dans une première étape, l'outil de construction sert à décrire l'ensemble des composants applicatifs. Cette description, appelée *modèle*, est stockée dans un fichier XML suivant la syntaxe Olan [BEL 00]. L'outil de configuration est ensuite utilisé pour définir l'instanciation de tous les composants qui seront initialement déployés sur les sites de l'application. Le résultat de cette phase de configuration est appelé une *configuration* qu'un outil de déploiement exploite pour installer les paquetages logiciels sur chaque site et activer l'exécution des composants.

Ce déploiement n'est possible que si l'infrastructure Middleware ScalAgent fonctionne sur chacun des sites. Cette infrastructure est qualifiée d'*homogène* du fait qu'elle a la même composition<sup>1</sup> sur les différents nœuds du système et qu'elle fournit donc les mêmes propriétés non fonctionnelles à tous les composants applicatifs. Seule l'intervention manuelle d'un spécialiste permet de configurer plus finement le middleware sur chaque site. Ceci est un inconvénient majeur lorsque certains sites aux capacités restreintes sont très sensibles aux performances et requièrent une configuration fine.

## 2.2. Architecture du middleware ScalAgent

Le middleware ScalAgent fournit un modèle de programmation de composants applicatifs adapté à la communication asynchrone et inspiré du modèle d'*acteurs* [AGH 86]. Chaque composant applicatif est créé et s'exécute au sein d'un représentant local du middleware appelé *serveur de composants*. Celui-ci assure la création des composants, leur exécution et leurs communications. Il héberge une fabrique de composants et un bus local.



**Figure 2.** Architecture d'un serveur de composants

Le bus local est composé de deux composants principaux (c.f. figure 2) :

- le composant *channel* qui transporte les messages de façon fiable et asynchrone en utilisant un système de files de messages.
- le composant *engine* qui provoque la réaction des composants applicatifs destinataires de messages. Il effectue, en boucle, un ensemble d'instructions consistant à prendre un message dans le *channel*, charger le composant applicatif destinataire du message et le faire réagir à ce message.

Il est, d'autre part, possible d'adjoindre aux deux composants principaux du bus à message certains composants appelés *composants middlewares*. Ceux-ci sont responsables de propriétés non fonctionnelles que l'on peut classer en deux catégories :

- les propriétés non fonctionnelles de composants telle que la persistance. Les composants middlewares responsables de telles propriétés sont localisés dans l'*engine*.

1. Le mot composition réfère à un ensemble de composants middlewares.

– les propriétés non fonctionnelles de connecteurs telle que la sécurité – qui consiste à chiffrer les messages émis sur un lien et à déchiffrer les messages reçus sur ce même lien – ou encore l’ordonnancement causal des messages échangés sur un ensemble de liens. Les composants middlewares responsables des propriétés non fonctionnelles de connecteurs sont localisés dans le *channel*.

Chaque composant middleware responsable d’une propriété non fonctionnelle possède un ensemble d’*attributs* qui lui permettent de fournir cette propriété. Le composant middleware *persistance* possède, par exemple, au moins deux attributs par composant applicatif auquel il fournit la propriété : le premier attribut est la technique de sauvegarde utilisée (disque dur, bande magnétique, base de données, etc.) tandis que le second attribut spécifie l’emplacement de la sauvegarde.

### 2.3. Vers une configuration de middleware dirigée par les applications

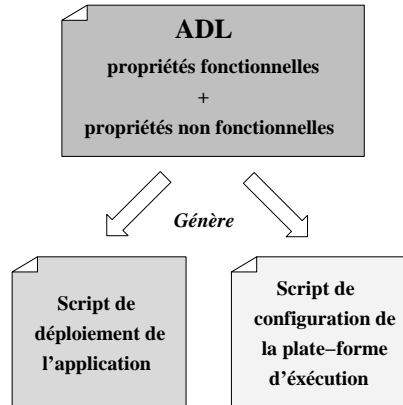
Comme nous l’avons mentionné dans la section 2.1, le middleware déployé par le processus de déploiement de l’application est homogène : tous les serveurs de composants hébergent les mêmes composants middlewares et fournissent, par conséquent, les mêmes propriétés non fonctionnelles. Le problème résultant de cette approche est que le middleware déployé offre souvent plus de fonctionnalités que nécessaires. Par exemple, il se peut que certains composants soient rendus persistants, alors même que l’application ne le requiert pas. Ceci peut être un inconvénient majeur pour des applications faisant intervenir des équipements aux ressources limitées.

Nous proposons de configurer les composants middlewares du MOM ScalAgent pour déployer, sur chaque site, uniquement ce qui est exigé par l’application. Nous proposons pour cela d’étendre l’ADL afin de permettre au développeur de spécifier les propriétés non fonctionnelles requises par son application. A partir de cette description étendue, un algorithme de configuration détermine la structure du middleware, c’est-à-dire l’ensemble des composants middlewares nécessaires sur les différents sites (c.f. figure 3).

## 3. Le langage de description d’architecture étendu

Les langages de description d’architecture ont pour objectif de fournir une vue structurée d’un système informatique. Ils sont basés sur des concepts communément acceptés [ISS 98b] : les *composants* définissent les entités de base du système ; les *connecteurs* définissent les types d’interaction entre les composants ; la *configuration* définit une architecture d’application en terme d’interconnexion de composants par l’intermédiaire de connecteurs.

Les ADL existant diffèrent par l’exploitation qu’ils font de la description de l’application. En effet, la recherche autour des ADL s’est concentrée sur deux points :



**Figure 3.** *Processus de construction d'une application*

– la génération d'un exécutable et son déploiement. C'est le cas d'ADL comme UniCon [SHA 95], Olan [BEL 00], Aster [ISS 98a] ou C2 [MED 99] qui utilisent la description de l'application pour automatiser son processus de déploiement.

– l'analyse du système. C'est le cas d'ADL tels que Rapide [LUC 95] ou Wright [ALL 97] qui permettent au développeur de l'application de spécifier le comportement dynamique des différentes entités du système. Ces ADL utilisent la description de l'application pour modéliser et analyser les scénarios envisageables.

L'ADL utilisé dans l'infrastructure ScalAgent est directement dérivé de l'ADL Olan et appartient donc à la première catégorie. A notre connaissance, il existe peu d'ADL de la première catégorie qui permettent de spécifier à la fois les propriétés non fonctionnelles des composants et celles des connecteurs. ACME [GAR 97] permet au développeur de spécifier ces deux types de propriétés. Cependant, il n'utilise pas cette description ; elle est uniquement mise à disposition d'autres outils. Le langage Aster [ISS 98a] propose de spécifier les propriétés non fonctionnelles de connecteurs à l'aide de la logique du premier ordre. Il ne traite que les propriétés non fonctionnelles relatives aux communications et nécessite du développeur la maîtrise d'un langage formel de spécification.

### 3.1. *Spécification des propriétés non fonctionnelles*

Nous proposons d'étendre la syntaxe Olan utilisée par les outils ScalAgent pour permettre au développeur de l'application de spécifier certaines propriétés non fonctionnelles requises par les composants ou les connecteurs. Olan permet de décrire ces deux entités en tant qu'éléments XML suivant une DTD décrite dans [BEL 00]. Suivant son type, une propriété non fonctionnelle peut donc être exprimée comme un

sous-éléments d'un élément composant ou d'un élément connecteur. La syntaxe utilisée pour déclarer une propriété non fonctionnelle est la suivante :

```
<property_name attribute_1="value" ...attribute_n="value"/>
```

Illustrons cette syntaxe avec le cas de la spécification de la propriété *persistance*. Il est important de pouvoir spécifier cette propriété car il peut être impossible de la fournir sur certains systèmes ne possédant pas de mémoire flash par exemple. Pour spécifier que le composant applicatif C1 a besoin d'être persistant et que son état doit être sauvegardé dans le fichier save/C1, le développeur emploie la syntaxe suivante :

```
<component name="C1">
    <persistence technique="file-system" uri="save/C1"/>
</component>
```

### 3.2. Spécification de la localisation des composants applicatifs

Les possibilités de spécification de la localisation des composants applicatifs ont aussi été étendues. La syntaxe Olan impose la spécification d'une et une seule localisation pour chaque composant d'une application. Nous permettons au développeur de spécifier, pour chaque composant applicatif, un ensemble (possiblement vide) de serveurs de composants qui peuvent héberger le composant. Notons que spécifier un ensemble vide signifie que le composant applicatif peut être localisé, a priori, sur n'importe quel serveur de composants, mais sa localisation dépendra, a posteriori, des propriétés non fonctionnelles demandées. La section suivante explique comment cette extension du pouvoir de spécification de l'ADL est utilisée pour déterminer la configuration du middleware qui répond le mieux aux besoins de l'application.

## 4. L'algorithme de configuration

### 4.1. Principes de l'algorithme

Le but de l'algorithme est de déterminer la localisation des composants applicatifs et l'ensemble des composants middlewares nécessaires sur chaque site pour satisfaire les besoins spécifiés à l'aide de l'ADL étendu. Etant donné que le middleware configuré doit être le plus efficace possible – c'est-à-dire doit entraîner un minimum de surcharge à l'exécution – il est nécessaire d'évaluer le coût de prise en charge des différentes propriétés. L'ensemble des coûts de prise en charge des différentes propriétés étant calculé, l'algorithme est capable de déterminer la localisation des composants applicatifs qui minimise le coût global de prise en charge des propriétés non fonctionnelles.

#### 4.2. *Evaluation du coût de gestion d'une propriété non fonctionnelle*

Chaque composant middleware responsable d'une propriété non fonctionnelle doit implémenter une fonction dont le but est de déterminer le coût de gestion de la propriété en fonction d'un certain nombre de paramètres. Cette fonction a la signature suivante : *int coutGestion(caractéristiquesMatériel, nombreComposants, nombreComposantsPropriété)* ;

Le premier paramètre correspond aux caractéristiques de l'équipement hébergeant le serveur de composants. Le deuxième paramètre est le nombre de composants applicatifs hébergés par le serveur de composants. Enfin, le troisième paramètre indique le nombre de composants parmi ceux hébergés par le serveur de composants qui nécessitent la propriété non fonctionnelle gérée par le composant middleware. La fonction retourne une valeur entière qui évalue le coût de prise en charge de la propriété non fonctionnelle pour le serveur de composant hébergeant ce composant middleware.

De plus, il peut exister plusieurs méthodes de prise en charge d'une propriété donnée, chacune engendrant un coût différent. Prenons l'exemple de la propriété non fonctionnelle *persistance*. Supposons qu'un serveur de composant héberge cinq composants applicatifs et que trois d'entre eux nécessitent d'être persistant. Le composant middleware responsable de la propriété a le choix entre deux méthodes de gestions :

1) gestion indépendante des différents composants applicatifs : il ne sauvegarde l'état que des trois composants qui requièrent la propriété.

2) gestion commune des différents composants applicatifs hébergés par un même serveur de composants : dès lors qu'un composant applicatif nécessite d'être persistant, tous les composants applicatifs hébergés par le serveur de composant sont rendus persistants. Dans l'exemple, les cinq composants sont rendus persistants.

Quand différentes méthodes de gestions sont envisageables, le choix se fait grâce à des fonctions d'évaluations de coût similaires à celle décrite précédemment. Par exemple, le développeur du composant middleware responsable de la persistance devra implémenter deux fonctions correspondants aux deux méthodes de gestion :

– *coutGestionIndépendante(caractéristiquesMatériel, nombreComposants, nombreComposantsPersistants)* qui permet d'évaluer le coût de gestion de la propriété dans le cas d'une gestion indépendante des différents composants applicatifs ;

– *coutGestionCommune(caractéristiquesMatériel, nombreComposants, nombreComposantsPersistants)* qui permet d'évaluer le coût de gestion de la propriété dans le cas d'une gestion commune des différents composants applicatifs hébergés par un même serveur de composant.

#### 4.3. *Choix de la localisation des composants*

Pour chaque placement envisageable des différents composants applicatifs, l'algorithme dispose des coûts de gestion des différentes propriétés non fonctionnelles.

Ces coûts ont été calculés grâce aux fonctions précédemment décrites et à la connaissance qu'a l'ADL des différents sites et de leurs caractéristiques matérielles. Il est donc possible de déterminer le placement des composants applicatifs qui minimise le coût global de prise en charge de l'ensemble des propriétés non fonctionnelles.

#### 4.4. Le langage de description du middleware

De façon similaire aux langages de description d'architecture (ADL), le langage de description de middleware (MDL pour *Middleware description language*) est un langage XML permettant de décrire la structure du middleware en terme de composants middlewares, leurs attributs et les relations entre eux. L'algorithme de configuration du middleware génère une description du middleware configuré à l'aide de ce langage. Cette description est utilisée par l'outil de déploiement pour installer la bonne configuration de middleware à chaque nœud. Prenons l'exemple d'un composant applicatif C1 qui requiert la propriété *persistance*. La figure 4 donne un exemple de ce que pourrait être la description du serveur de composants hébergeant C1.

```
<serverConfiguration sid="1">
  <persistenceConfiguration management="independent">
    <component name="C1" technique="file system" file="..." />
  </persistenceConfiguration>
</serverConfiguration>
```

**Figure 4.** Exemple de fichier de description du serveur de composants hébergeant C1

Ce fichier XML indique que le serveur de composants identifié par le sid 1 possède le composant middleware responsable de la persistance. Il est aussi précisé que la gestion de la propriété est faite de façon indépendante pour les différents composants et que le seul composant hébergé par ce serveur de composants nécessitant d'être persistant est le composant C1. Cette description du composant middleware responsable de la persistance donne aussi les attributs nécessaires au composant pour sauvegarder l'état du composant C1 : la technique et le fichier de sauvegarde.

## 5. Evaluation des performances

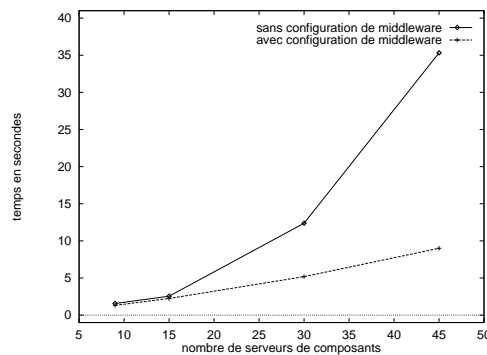
Cette section montre, qu'outre faciliter le développement d'applications réparties, l'algorithme de configuration proposé permet d'améliorer sensiblement les performances des applications en réduisant le surcoût engendré, par le middleware, pour la prise en charge des propriétés non fonctionnelles nécessaires à l'application.

### 5.1. Protocole de test

Des mesures de performances ont été réalisées sur des applications pour lesquelles le middleware ordonne causalement les messages échangés par les composants appli-

catifs. L'indicateur de performance mesuré est le temps moyen mis par un message pour parvenir d'un composant applicatif à un autre. Deux méthodes d'ordonnement causal sont comparées : la première est basée sur l'utilisation d'horloges matricielles ; la seconde, basée sur un protocole décrit dans [QUE 02], nécessite une phase d'analyse effectuée par l'algorithme de configuration. Afin de pouvoir effectuer des mesures à grandes échelles, nous avons utilisé une grappe de PC<sup>2</sup>. Nous ne présentons que les résultats principaux. D'autres mesures sont disponibles dans [QUE 02].

### 5.2. Mesures et résultats



**Figure 5.** Application avec composants applicatifs moyennement connectés

La figure 5 montre très clairement le gain de performance obtenu avec le middleware configuré à l'aide de l'algorithme présenté au paragraphe 4. Ces résultats ont été obtenus avec le MOM ScalAgent. Cependant, nous pensons qu'ils sont suffisamment généraux pour être appliqués à tout middleware utilisant une gestion matricielle de l'ordonnement causal des messages échangés.

## 6. Conclusion

De nos jours, les applications réparties font intervenir des systèmes hétérogènes allant des ordinateurs traditionnels à des équipements mobiles aux ressources très limitées. La construction, la configuration et le déploiement de telles applications sont des tâches très difficiles du fait de nombreux facteurs : dispersion géographiques des différents équipements, nombre important d'entités impliquées dans l'application, hétérogénéité des plates-formes, etc. L'infrastructure ScalAgent a pour but de faciliter ces tâches en proposant un ensemble d'outils basés sur un middleware orienté messages et sur une approche orientée composant de la construction des applications.

2. Les PC utilisés pour les mesures sont des Pentium III cadencés à 700 MHz avec 256 Mo de mémoire, connectés par un réseau Ethernet à 100 Mbit/s. Le système d'exploitation utilisé est Linux 2.4.6.

Cependant, les applications modernes sont de plus en plus concernées par l'augmentation du nombre d'équipements mobiles ou embarqués à ressources limitées. De fait, les applications doivent être capables d'intégrer ces nouveaux équipements. Pour ce faire, le middleware doit être configurable de façon à faire face au manque de ressources de ces équipements, à leur diversité, tout en fournissant aux applications une interface identique quel que soit l'équipement.

Nous avons présenté, dans cet article, les extensions apportées à la plate-forme ScalAgent pour qu'elle supporte l'hétérogénéité des équipements et les besoins applicatifs. Les capacités de configuration du middleware ScalAgent ont été augmentées de façon à pouvoir configurer indépendamment les différents sites du système et ce, automatiquement sans intervention humaine. Les outils de description d'applications ont été étendus de façon à pouvoir spécifier les propriétés non fonctionnelles nécessaires à l'application. A partir de cette description globale de l'application, un outil de déploiement permet de construire automatiquement une plate-forme d'exécution répondant aux besoins de l'application. Cet outil est piloté par un algorithme de configuration qui essaie de réduire le coût de gestion des différentes propriétés non fonctionnelles. Pour ce faire, l'algorithme détermine le placement des composants applicatifs qui minimise le coût de gestion des propriétés non fonctionnelles.

**Travaux connexes.** De nombreux travaux ont été consacrés à la configuration des middlewares au cours des dix dernières années. Le projet Aster [ISS 98a] propose d'utiliser des méthodes formelles basées sur la logique du premier ordre pour étudier la concordance entre un bus logiciel offrant certaines fonctionnalités et des besoins applicatifs. L'inconvénient d'une telle approche est qu'il s'avère très complexe d'exprimer formellement le comportement du bus logiciel et les besoins des applications. Le projet Lasagne [TRU 01] propose de faire une composition d'aspects [KIC 97] à l'exécution pour configurer dynamiquement le système. Cette composition d'aspect est effectuée à l'aide d'un tisserand (weaver) qui utilise des techniques réflexives pour effectuer une sélection des aspects en fonction du contexte d'exécution. Selon les concepteurs de Lasagne, l'inconvénient majeur de cette technique est la surcharge causée à l'exécution par le tisserand. Des travaux antérieurs ont aussi été menés dans le but de configurer des protocoles de communications [AST 95]. Cependant, cette proposition ne prend pas en compte les besoins non fonctionnels de l'application.

#### Remerciements

Les auteurs tiennent à remercier Roland Balter et Philippe Laumay pour leurs conseils avisés sur ce travail et leurs précieux commentaires sur cet article.

## 7. Bibliographie

[AGH 86] AGHA G. A., « Actors : A Model of Concurrent Computation in Distributed Systems », *The MIT Press, ISBN 0-262-01092-5*, Cambridge, MA, 1986.

- [ALL 97] ALLEN R., GARLAN D., DOUENCE R., « Specifying Dynamism in Software Architectures », *Proceedings of the Workshop on Foundations of Component-Based Software Engineering*, Zurich, Switzerland, September 1997.
- [AST 95] ASTLEY M., STURMAN D. C., AGHA G. A., « Customizable Middleware for Modular Distributed Software », *Communications of the ACM*, 1995.
- [BEL 99] BELLISSARD L., PALMA N. D., FREYSSINET A., HERRMANN M., LACOURTE S., « An Agent Platform for Reliable Asynchronous Distributed Programming », Symposium on Reliable Distributed Systems, October 1999.
- [BEL 00] BELLISSARD L., PALMA N. D., FÉLIOT D., *The Olan Architecture Definition Language*, C3DS Technical Report, volume 24, 2000.
- [GAR 97] GARLAN D., MONROE R. T., WILE D., « Acme : An Architecture Description Interchange Language », *Proceedings of CASCON'97*, Toronto, Ontario, November 1997, p. 169-183.
- [ISS 98a] ISSARNY V., BIDAN C., SARIDAKIS T., « Achieving Middleware Customization in a Configuration-Based Development Environment : Experience with the Aster Prototype », *Proceedings of the 4th International Conference on Configurable Distributed Systems*, Annapolis, Maryland, USA, May 1998, p. 207-214.
- [ISS 98b] ISSARNY V., SARIDAKIS T., ZARRAS A., *A Survey of Architecture Description Languages*, C3DS Deliverable A3.1, ESPRIT LTR Project N°24962, 1998.
- [KIC 97] KICZALES G., LAMPING J., MENDHEKAR A., MAEDA C., LOPES C. V., LOINGTIER J.-M., IRWIN J., « Aspect-Oriented Programming », *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, June 1997.
- [LUC 95] LUCKHAM D. C., KENNEY J. J., AUGUSTIN L. M., VERA J., BRYAN D., MANN W., « Specification and Analysis of System Architecture Using Rapide », *IEEE Transactions on Software Engineering, Special Issue on Software Architecture, Vol. 21, No. 4*, April 1995, p. 336-355.
- [MED 99] MEDVIDOVIC N., ROSENBLUM D. S., TAYLOR R. N., « A Language and Environment for Architecture-Based Software Development and Evolution », *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, 1999, p. 44-53.
- [QUE 02] QUEMA V., « Configuration d'un Middleware Dirigée par les Applications », Master's thesis, École Doctorale Mathématique et Informatique - Grenoble, June 2002, [http://sardes.inrialpes.fr/~quema/rapport\\_DEA.pdf](http://sardes.inrialpes.fr/~quema/rapport_DEA.pdf).
- [SHA 95] SHAW M., DELINE R., KLEIN D. V., ROSS T. L., YOUNG D. M., ZELESNIK G., « Abstractions for Software Architecture and Tools to Support Them », *Software Engineering*, vol. 21, n° 4, 1995, p. 314-335.
- [TRU 01] TRUYEN E., VANHAUTE B., JOOSEN W., VERBAETEN P., JØRGENSEN B. N., « Dynamic and Selective Combination of Extensions in Component-Based Applications », *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, Toronto, Canada, May 2001.