Une architecture de composants répartis adaptables

An Phung-Khac, Jean-Marie Gilliot, et Maria-Teresa Segarra

Département Informatique, TELECOM Bretagne
Technopôle Brest-Iroise, CS83818 29238 Brest cedex 3, France
{an.phungkhac,jm.gilliot,mt.segarra}@telecom-bretagne.eu

Résumé. Plusieurs travaux récents proposent des solutions ou des frameworks dédiés au développement d'applications adaptables qui peuvent ainsi dynamiquement changer leur comportement pendant l'exécution afin de s'adapter au contexte d'exécution courant. Cependant, avec ces approches, les tâches à la charge des développeurs sont encore complexes. Ces tâches incluent la définition des variantes et la spécification des actions d'adaptation, qui dans le contexte des systèmes répartis, incluent des contraintes liées aux parties distribuées. Dans cet article, nous présentons une approche de développement d'applications réparties adaptables permettant la génération correcte des variantes d'une application et des actions d'adaptation à exécuter en vue de faciliter la tâche des développeurs.

1 Introduction

Les applications informatiques sont omniprésentes dans notre environnement. Celles-ci doivent désormais pouvoir fonctionner dans des contextes très variables, même si ceci requiert la modification de leurs structures internes sans même s'interrompre. Le contexte comprend les dispositifs, les réseaux, le comportement de l'environnement et de l'utilisateur. On parle d'adaptation dynamique.

Plusieurs travaux récents proposent des solutions ou des frameworks dédiés au développement de telles applications adaptables qui peuvent dynamiquement changer leur comportement pendant l'exécution afin de s'adapter au contexte d'exécution courant. Cependant, avec ces approches, les tâches de développement d'une application répartie adaptable à la charge des développeurs sont encore complexes. Cette complexité est dûe notamment aux deux tâches suivantes :

- *Spécification des variantes de l'application* : Lors d'une session d'adaptation, l'application doit changer d'une variante cohérente à une autre. Dans le cas des applications réparties, une telle variante se compose de parties distribuées qui collaborent pour offrir des fonctions particulières. La spécification de telles variantes peut alors s'avérer une tâche difficile.
- *Spécification des transitions entre variantes* : les transitions d'adaptation entre variantes sont aussi complexes. Chaque transition comprend des actions de changement d'architecture, de configuration et/ou de paramètre. En plus, lorsque l'application gère des données, il est important que celles-ci soient transférées à la nouvelle variante. Ce transfert

est d'autant plus complexe que les applications sont distribuées, les actions devant alors être réparties.

Dans cet article, afin de faciliter ces tâches, nous présentons une approche de développement d'applications réparties adaptables grâce à laquelle les variantes architecturales d'une application et des actions d'adaptation à réaliser sont correctement et automatiquement générées.

Dans notre approche, une application répartie est d'abord spécifiée au niveau abstrait comme l'interconnexion d'un ensemble de composants clients et d'un composant de communication (le composant de communication est aussi appelé *médium*). Puis, par un processus de raffinement, cette spécification abstraite est transformée en plusieurs variantes d'implantation. Celles-ci sont ensuite regroupées dans un nouveau médium que nous appelons *médium adaptable* et des contrôleurs d'adaptation y sont greffés qui sont en charge d'effectuer les changements d'une variante à une autre. Le médium adaptable ayant une structure clairement définie et connue, la planification des actions d'adaptation peut être automatisée.

L'article est organisé de la manière suivante. La section 2 introduit la notion de composant de communication ou médium sur laquelle repose notre proposition. Nous utilisons cette architecture dans notre approche de développement de médiums adaptables qui sera montré dans la section 3. Ensuite, la section 4 discute des travaux connexes. Enfin, la section 5 conclut l'article.

2 Médium

2.1 Définition

La notion de médium a été proposée au sein de notre équipe dans Cariou et al. (2002). Un composant de communication (ou *médium*) est un composant logiciel dans le sens où il est défini par une interface de services offerts et requis. La particularité de ce composant réside dans le fait qu'il est dédié à la communication et donc qu'il a par nature une mise en œuvre répartie. Un composant de communication n'est pas une unité de déploiement. C'est une unité architecturale logique construite pour être distribuée. Une application est le résultat de l'interconnexion d'un ensemble de composants clients et de composants de communication. Cette particularité des composants de communication présente un atout majeur dans le sens où elle permet de séparer dans une application la définition de l'aspect fonctionnel (dans les composants clients) de la définition de l'aspect communication (dans les composants de communication) Cariou et al. (2002).

2.2 Exemple : médium de réservation

À titre d'illustration, nous reprenons et simplifions l'exemple publié dans Cariou et al. (2002) sur une compagnie aérienne possédant un siège et des agences localisées sur des sites distants. Le problème consiste à attribuer un identificateur de réservation ou numéro de siège d'un avion à une personne souhaitant effectuer une réservation pour un vol donné. L'ensemble des identificateurs de réservation disponibles (*available* dans la figure 1b) est initialisé par le siège de la compagnie. Les identificateurs de réservation peuvent être gérés par un composant de communication qui offre par exemple, un service *setReserveIdSet* pour initialiser les identificateurs, un service *reserve* pour effectuer une réservation et un service *cancel* pour

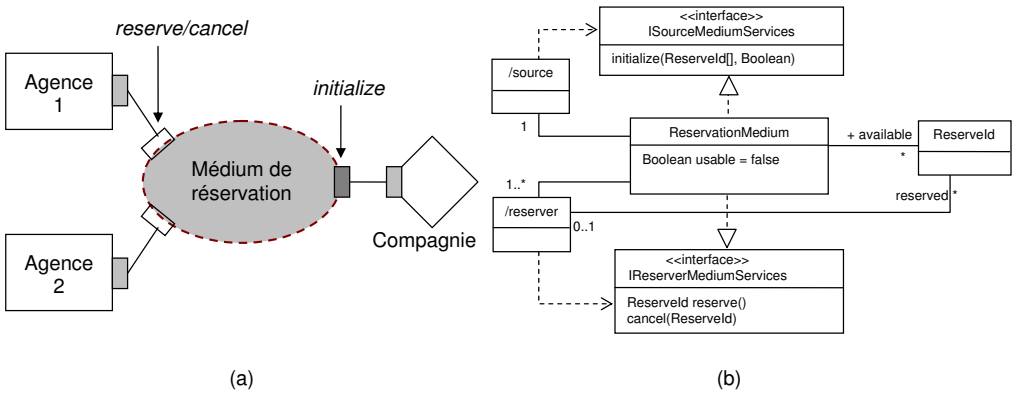

FIG. 1 – *Spécification abstraite du médium de réservation*

annuler une réservation. On peut construire une application de réservation en interconnectant ce composant de communication avec les composants représentant le siège et les agences de la compagnie aérienne comme illustré sur la figure 1a. Dans ce cas, pour effectuer (respectivement annuler) une réservation, chaque agence se connecte au composant et utilise le service *reserve* (respectivement *cancel*). De même, pour initialiser les identificateurs, le siège de la compagnie se connecte au composant et utilise le service *setReserveIdSet*.

2.3 Répartition, déploiement

Au niveau abstrait, un composant de communication est représenté par une seule entité logique. Cette entité logique n'est pas une unité de déploiement. Lors du déploiement, il est éclaté en plusieurs éléments appelés gestionnaires de rôle (ou *Managers*). Chacun des gestionnaires de rôle est associé à un composant client et est déployé avec lui sur le même site (figure 2). Chaque gestionnaire de rôle implante les services offerts par le composant de communication au composant client auquel il est associé et communique avec les autres gestionnaires de rôle pour assurer le service de communication. Le composant de communication est alors défini par l'agrégation logique des gestionnaires de rôle.

Les gestionnaires de rôle gèrent aussi les données du médium. Plusieurs stratégies ou algorithmes différents (Chord Stoica et al. (2001), Pastry Rowstron et Drusche (2001), ...) peuvent être utilisés pour implanter la gestion des données du composant de communication par les gestionnaires de rôle. Chaque stratégie aboutit à une implantation spécifique (centralisée, distribuée, répartie, mixte, etc.) du composant de communication. Plusieurs variantes d'implantation réparties peuvent donc exister pour l'architecture de déploiement d'un composant de communication. Dans Kaboré et Beugnard (2007), un processus de raffinement de médiums, de sa spécification abstraite aux variantes d'implantation de l'architecture (plus de 10 variantes) a été présenté. Ce processus se compose de plusieurs étapes correspondant à des préoccupations séparées au début du processus. A chaque étape, des choix de conception sont

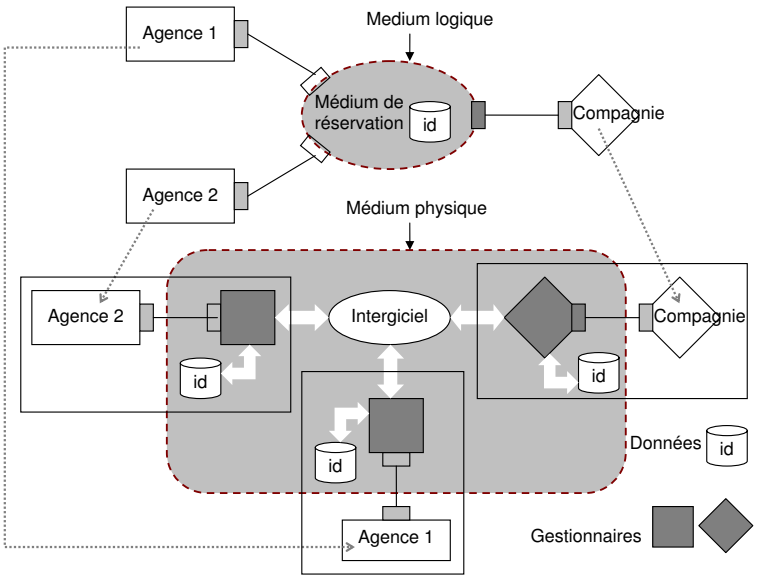

FIG. 2 – *Apperçu de l'architecture de déploiement du médium de réservation*

introduits. De cette façon, le processus de raffinement assure la cohérence fonctionnelle des parties distribuées de chaque variante architecturale du médium.

3 Médium adaptable

3.1 Approche proposée

Nous avons utilisé la notion de médium (Cariou et al. (2002)) et le processus de raffinement de médium (Kaboré et Beugnard (2007, 2008)) pour construire des applications réparties adaptables. La figure 3 montre le principe de notre approche :

- *Spécification abstraite :* Afin de développer une application répartie adaptable, nous utilisons le médium pour spécifier l'application. Elle est donc spécifiée comme l'interconnexion d'un médium et des composants fonctionnels.
- *Génération des variantes architecturales :* A partir de la spécification abstraite, nous utilisons le processus de raffinement pour transformer cette spécification abstraite en des spécifications d'implantation correspondant aux choix de conception introduits. Les choix de conception sont identifiés grâce à des préoccupations (Kaboré et Beugnard (2007)).
- *Spécification des transferts des données entre les variantes :* Le processus de raffinement se compose de plusieurs étapes qui raffinent pas à pas la spécification abstraite au niveau d'implantation en plusieurs variantes. Pendant ce processus, nous spécifions aussi les

actions de transfert des données entre ces variantes en identifiant des fonctions à utiliser pour récupérer et distribuer ces données.

– *Composition :* Les variantes architecturales sont composées dans une architecture de médium adaptable appelée *adapt-medium*. Cette architecture est constituée de deux parties : fonctionnelle et adaptative. La partie fonctionnelle contient les variantes architecturales dont une est activée pendant l'exécution. La partie adaptative correspond à un contrôleur d'adaptation qui utilise la spécification des transitions pour générer et exécuter des plans d'adaptation afin de changer la variante activée et transférer des données. L'application répartie adaptable est donc constituée du médium adaptable et les composants clients. Elle peut dynamiquement changer la variante de médium lorsque le contexte change.

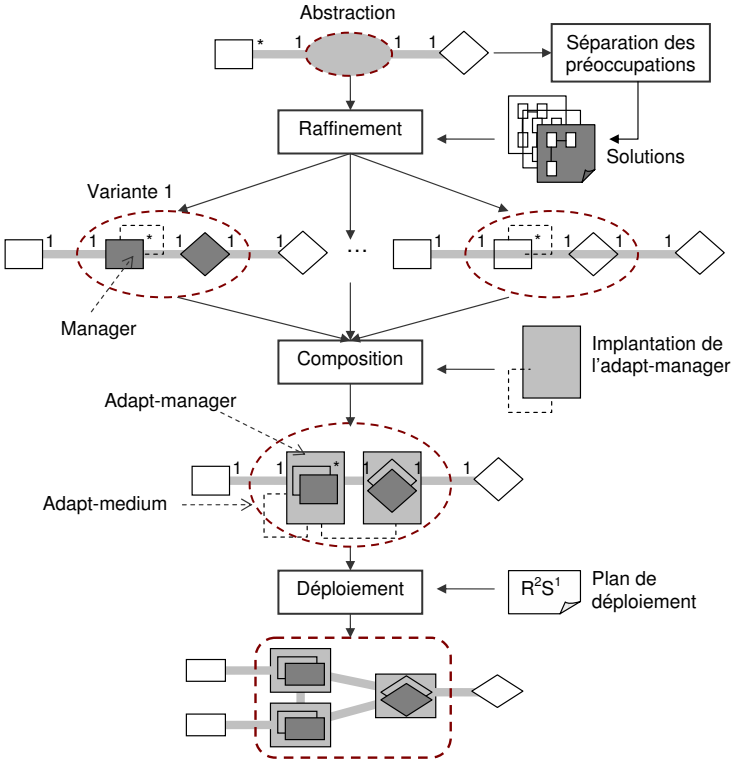

FIG. 3 – *Notre approche de développement d'applications réparties adaptables*

Dans les sections suivantes, nous précisons comment la génération des variantes, la spécification des transitions et la composition sont réalisées sur un exemple de médium adaptable de réservation. Ce médium peut être utilisé dans des applications de réservation qui peuvent dynamiquement changer de variante de médium afin de s'adapter au contexte.

3.2 Génération des variantes architecturales

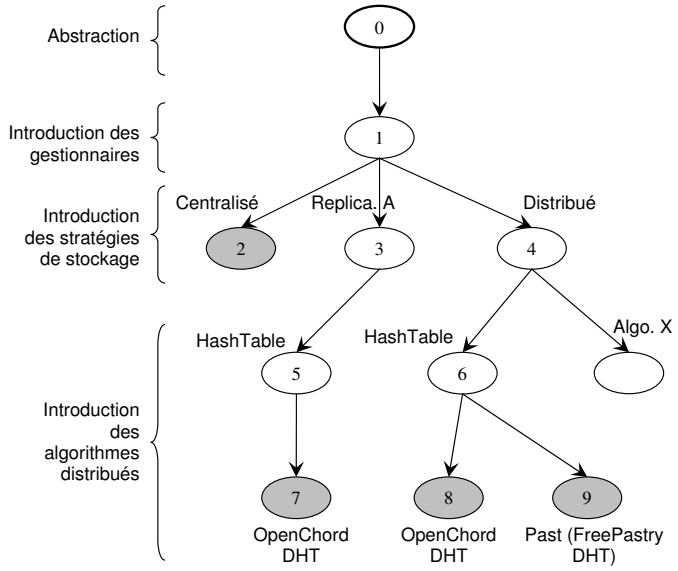

FIG. 4 – *Processus de raffinement du médium de réservation*

Le processus de raffinement se compose de plusieurs étapes qui transforment pas à pas la spécification abstraite en plusieurs spécifications d'implantation en introduisant des choix de conception. La figure 4 montre le processus de raffinement du médium de réservation. Comme le montre la figure, la spécification abstraite (le nœud 0) est raffinée en quatre variantes d'implantations (les nœuds 2, 7, 8, 9).

Le nœud 1 correspond à la spécification du médium où les gestionnaires de rôle sont introduits. Comme le montre la figure 5, un gestionnaire est associé à un composant client, les gestionnaires offrent alors les services du médium comme les portes de communication des composants clients.

Dans cette étape, les données (l'ensemble des identificateurs disponibles *available*) sont encore gérées par une partie abstraite du médium (la classe *ReservationMedium*). Le problème majeur dans la définition d'une implantation répartie d'un médium porte sur la gestion de la répartition de ces données. Pour faciliter cette définition, nous décomposons la gestion de la répartition de l'ensemble *available* en deux préoccupations successives :

- Stratégie de stockage de données : Les identificateurs disponibles peuvent être stockés de manière centralisée chez un gestionnaire *ReserverManager* (nœud 2), répliquée sur un ensemble des gestionnaires *ReserverManager* (nœud 3), ou distribué sur les gestionnaires *ReserverManager* (nœud 4).
- Algorithme réparti : Les identificateurs disponibles sont partagés entre les gestionnaires *ReserverManager*. Lorsqu'un gestionnaire ne gère pas l'identificateur requis, il a besoin d'un algorithme pour le trouver. Les algorithmes peuvent être classifiés par le type

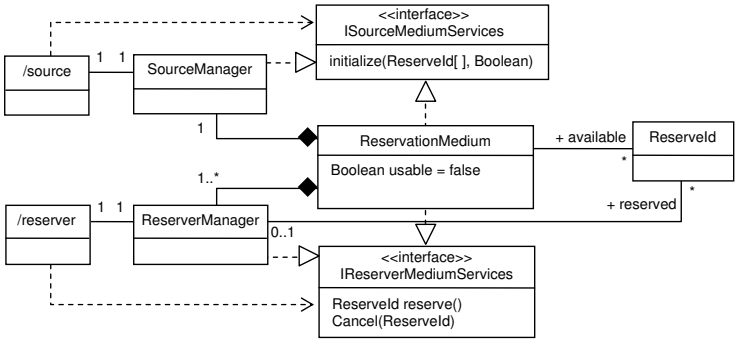

FIG. 5 – *Introduction des gestionnaires de rôle*

de données d'indexation utilisé. Par exemple lorsque les identificateurs sont distribués ou répliqués, OpenChord (Bamberg University, Distributed System Group (2007)) qui implante l'algorithme Chord (Stoica et al. (2001)) peut être utilisé.

La figure 6 indique la spécification correspondant au nœud 6. Les gestionnaires de données distribuées *SourceDistributedDataManager* et *ReserverDistributedDataManager* sont introduits. Ils implantent la stratégie distribuée de stockage de données. L'ensemble des identificateurs disponibles est maintenant distribué sur les gestionnaires de rôle *ReserverManager* comme des ensembles *localAvailable*. Pour un gestionnaire de rôle *ReserverManager*, un gestionnaires de données *ReserverDistributedDataManager* est le proxy de l'ensemble *available*.

Dans cette étape, le type de données *HashTable* a été choisi pour l'indexation des identificateurs disponibles. Ces données *hashTable* sont encore gérées par la partie abstraite *ReservationMedium* du médium.

L'étape suivante du raffinement permet de générer les nœuds 8 et 9. Dans cette étape, les objets des algorithmes sont introduits, la partie abstraite du médium disparaît, les spécifications sont complètement au niveau d'implantation.

3.3 Spécification du transfert de données entre les variantes

La génération automatique des plans d'adaptation pour un médium adaptable requiert l'identification des actions d'adaptation sur l'architecture du médium et des actions pour le transfert de données entre variantes. Les permières peuvent être obtenues à partir des étapes du processus de raffinement décrit dans le paragraphe précédent. Pour les deuxièmes, nous identifions les actions élémentaires de transfert de données entre deux nœuds connexes dans l'arbre de raffinement. Les actions comprennent les services du médium à utiliser pour transférer ces données. Par exemple, pour la chaîne des choix de conception de la spécification abstraite (nœud 0) à la variante correspondant au noeud 8 (appelée V8), les actions élémentaires identifiées sont les suivantes :

```
(0) -> (1):
  // Il n'y a pas de donnée à transféfer
(1) -> (4):
```

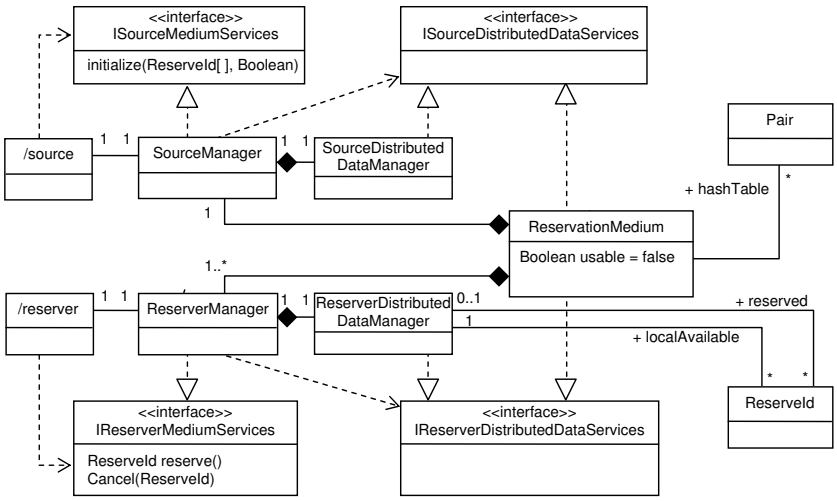

FIG. 6 – *Introduction de stratégies de stockage de données*

```
  // distribution de l'<available>
  for ID in (1).available
  {
    (4).SourceManager.insert(ID)
    // le service <insert> est implanté dans <SourceManager>
  }
(4) -> (1):
  // restaurer l'<available>
  (1).available = null
  for m in {all managers}
  {
    (1).available.add(m.localAvailable)
  }
(4) -> (6):
  // Il n'y a pas de donnée à transréfer
(4) -> (8):
  // Construire le table de hachage pour <OpenChord>
  for m in {all managers}
  {
    for ID in m.localAvailable
    {
      m.ChordObject.add(ID,m.name)
    }
  }
(8) -> (4):
  // Il n'y a pas de donnée à restaurer
```

Pour la variante correspondant au nœud 2 (appelée V2), les actions élémentaires identifiées sont les suivantes :

```
(0) -> (1):
  // Il n'y a pas de donnée à transférer
(1) -> (2):
  (2).serverNode.available = (1).available
(2) -> (1):
  (1).available = (2).serverNode.available
```

En utilisant ces actions élémentaires, nous pouvons générer les actions de transfert des données entre ces deux variantes. Par exemple, pour transférer les données de la variante V2 (centralisée) à la variante V8 (distribué, OpenChord), le chemin est (2)->(1)->(4)->(8) et les actions nécessaires de transfert sont :

```
variante V2 à variante V8 : {
  // (2) -> (1) :
  (1).available = (2).serverNode.available
  // (1) -> (4):
  for ID in (1).available
  {
    (4).SourceManager.insert(ID)
  }
  // (4) -> (8):
  for m in {all managers}
  {
    for ID in m.localAvailable
    {
      m.ChordObject.add(ID,m.name)
    }
  }
}
```

Ces actions élémentaires sont composées avec les variantes architecturales dans un médium adaptable où les actions de transfert de données sont générées dynamiquement.

3.4 Composition

Les variantes architecturales et les actions élémentaires de transfert de données sont intégrées dans un médium adaptable appelé *adapt-medium*. Ce médium adaptable est une agrégation logique de gestionnaires de rôle adaptables (*adapt-managers*), un par composant client. Nous avons implanté les *adapt-managers* en utilisant le canevas d'adaptation DYNACO Buisson et al. (2005) qui separe les méchanismes d'adaptation en un décideur, un planificateur, et un exécuteur.

Comme le montre la figure 7, chaque gestionnaire de rôle adaptable se compose d'un composite (*CompositeManager*) qui encapsule les variantes possibles pour le gestionnaire, d'un contrôleur d'adaptation (*AdaptationController*) et d'un composant *MediumLogic*. Le *CompositeManager* contient les variantes du gestionnaire de rôle associé à un même composant client.

Par exemple, la figure 8 montre l'architecture du gestionnaire de réservation pour les variantes V2 et V8. Ces variantes seront intégrées dans le *adapt-manager* associé au client de réservation. En ce qui concerne le *AdaptationController*, il est constitué de trois composants : le *decider* décide du moment de lancer une session d'adaptation et la variante à adopter, le *planner* utilise les actions élémentaires pour générer le plan d'adaptation à exécuter (transformation architecturale et transfert de données) et le *executor* qui réalise effectivement les actions. Le composant *MediumLogic* gère l'architecture globale des variantes du médium adaptable.

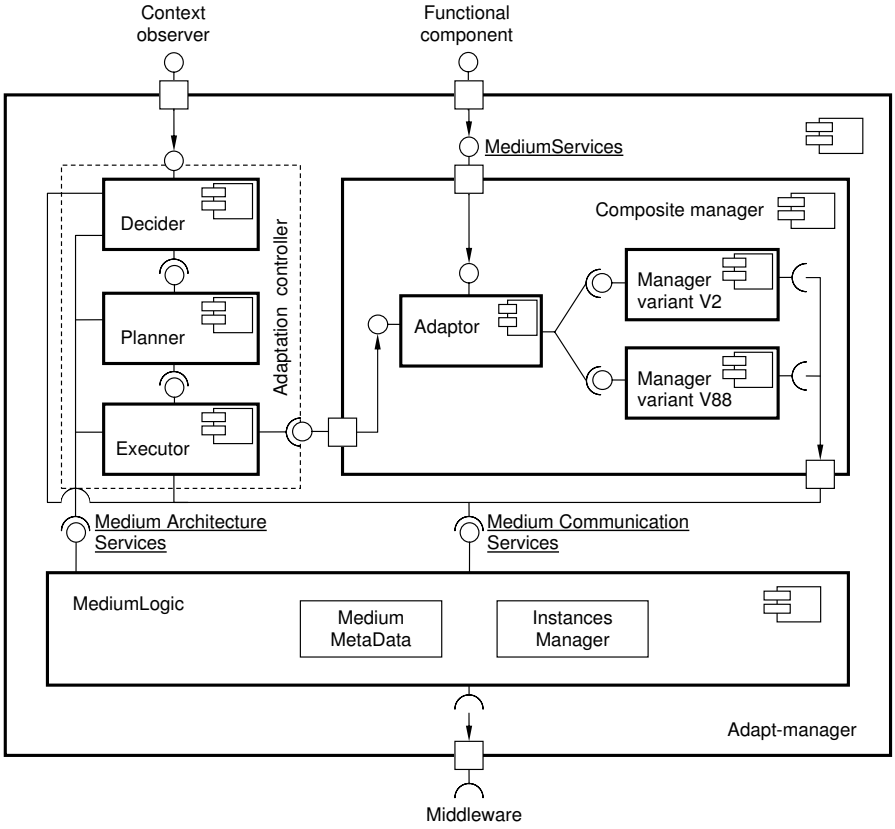

FIG. 7 – *Architecture du composant* adapt-manager

Le processus de raffinement ainsi que la composition pour obtenir un médium adaptable peuvent être automatisés par des transformations de modèles (Kaboré et Beugnard (2007); Phung-Khac et al. (2008)).

4 Travaux connexes

Plusieurs travaux récents proposent des solutions ou des frameworks dédiés au développement d'applications adaptatives qui peuvent dynamiquement changer leur comportement afin

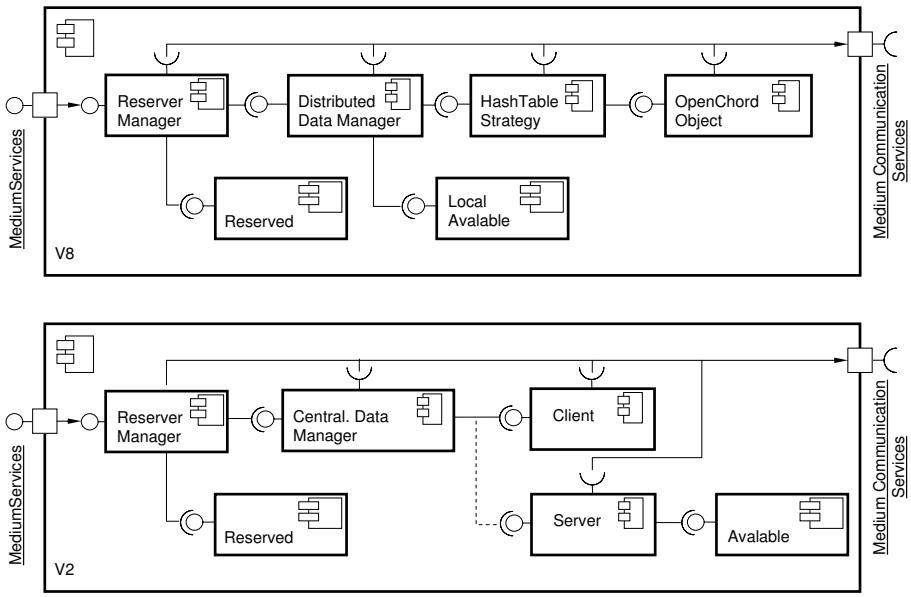

FIG. 8 – *Modèles de composants des gestionnaires des variantes V2 et V8*

de s'adapter au contexte courant. Mais, à notre connaissance, il n'existe par de solutions permettant la planification automatique des adaptations distribuées.

Dans le contexte des applications adaptables basées sur des composants, Buisson et al. (2005); Chefrour (2005); Kon et al. (2005) ont proposé des plateformes spécialisables qui offrent des mécanismes d'adaptation/reconfiguration. Ces travaux se concentrent sur la structure des méchanismes d'adaptation des applications. Avec ces approches les tâches de planification de l'adaptation sont prises en charge par les développeurs lors du développement.

Dans Ayed et Berbers (2007); Geihs et al. (2006), les auteurs ont proposé des approches de composition d'applications adaptatives. Cependant, ces approches ne sont pas applicables pour les systèmes ayant des besoins d'adaptation distribuée.

Dans le domaine de la robotique, certains travaux proposent des solutions pour la planification automatique des actions d'adaptation. Par exemple, Sykes et al. (2008) a proposé un modèle en trois couches où les plans d'adaptation sont générés à partir des modèles de but exprimés en logique temporelle. Cependant, les plans n'impliquent pas des applications distribuées.

Dans Bencomo et al. (2008), les auteurs proposent une approche de modélisation de variantes architecturales des applications basées sur les composants. Mais ces variantes ne sont pas distribuées.

5 Conclusion

Nous avons présenté notre approche de développement d'applications réparties adaptables qui peut simplifier les tâches de construction des variantes des applications et de planification

des transitions lors d'une adaptation.

Dans notre approche, une application distribuée est d'abord spécifiée au niveau abstrait comme l'interconnexion d'un ensemble de composants client et d'un composant de communication ou médium. Puis, par un processus de raffinement réifié par des transformations de modèles, cette spécification est transformée dans plusieurs variantes d'implantation. Chacune des variantes est constituée d'un gestionnaire de rôle par composant client, chaque type de composant client ayant un type de gestionnaire de rôle. Pour chacun de ces types, nous regroupons les variantes d'implantation qui ont été obtenues par le processus de raffinement. Enfin, nous introduisons un contrôleur d'adaptation par gestionnaire de rôle qui s'occupe de gérer le changement dynamique de la variante courante. L'application résultant de l'interconnexion des composants clients et de notre médium adaptatif peut ainsi changer l'architecture de communication afin de satisfaire les besoins exprimés.

Notre médium adaptatif comportant les différentes variantes et les plans de transfert de données, la session d'adaptation peut être menée en trois étapes : 1) démarrage de la nouvelle variante, 2) transfert des données de l'ancienne variante à la nouvelle et 3) changement de variante active.

Le fait de nous appuyer sur un processus basé sur des modèles nous permet effectivement de réaliser de manière automatique la coordination des changements de variantes et les plans de transfert des données. Le concepteur de telles applications est ainsi déchargé d'un des aspects les plus complexes de la construction et peut alors se concentrer sur la spécification des variantes souhaitées.

Enfin, nos travaux se sont focalisés sur l'automatisation de la planification et exécution d'adaptations distribuées. Notre hypothèse de travail a été, donc, l'existence d'une fonction de décision Buisson et al. (2005) qui déclenche les adaptations lorsque ceci est nécessaire. L'ajout d'une telle fonction dans notre approche requiert d'enrichir le processus de raffinement avec des informations sur le(s) contexte(s) d'exécution pertinent(s) pour chaque variante d'implantation et d'intégrer dans le médium adaptatif une fonctionnalité de décision exploitant cette information.

Références

Ayed, D. et Y. Berbers (2007). Dynamic adaptation of CORBA component-based applications. In *Proceedings of the 2007 ACM symposium on Applied Computing (SAC'07)*, pp. 580–585. ACM Press.

Bamberg University, Distributed System Group (2007). Openchord. http ://www.uni-bamberg.de/projects/openchord.

Bencomo, N., G. Blair, C. Flores, et P. Sawyer (2008). Reflective Component-based Technologies to Support Dynamic Variability. In *2nd Workshop on Variability Modelling of Software-intensive Systems (VaMoS08)*, Germany.

Buisson, J., F. André, et J.-L. Pazat (2005). A framework for dynamic adaptation of parallel components. In *ParCo 2005*.

Cariou, E., A. Beugnard, et J.-M. Jézéquel (2002). An architecture and a process for implementing distributed collaborations. In *Proceedings of the 6th IEEE International Enterprise*

Distributed Object (EDOC 2002), Lausanne, Switzerland, pp. 132–143. IEEE Computer Society.

Chefrour, D. (2005). Developing component-based adaptive applications in mobile environments. In *SAC '05 : Proceedings of the 2005 ACM symposium on Applied computing*, New York, NY, USA, pp. 1146–1150. ACM Press.

Geihs, K., M. U. Khan, R. Reichle, A. Solberg, S. Hallsteinsen, et S. Merral (2006). Modeling of component-based adaptive distributed applications. In *Proceedings of the 2006 ACM symposium on Applied Computing (SAC'06)*, pp. 718–722. ACM Press.

Kaboré, E. et A. Beugnard (2007). Automatisation d'un processus. *RSTI - L'Objet 13/1007*, 105–135.

Kaboré, E. et A. Beugnard (2008). Implementing a Data Distribution Variant with a Metamodel, Some Models and a Transformation. In *Proceeding of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08)*, Lecture Notes in Computer Science, Oslo, Norway. Springer-Verlag. to appear.

Kon, F., J. R. Marques, T. Yamane, R. H. Campbell, et M. D. Mickunas (2005). Design, implementation, and performance of an automatic configuration service for distributed component systems : Research articles. *Softw. Pract. Exper. 35*(7), 667–703.

Phung-Khac, A., A. Beugnard, J.-M. Gilliot, et M.-T. Segarra (2008). Model-Driven Development of Component-based Adaptive Distributed Applications. In *Proceeding of the 23rd ACM Symposium on Applied Computing (SAC'2008), track on Dependable and Adaptive Distributed Systems (DADS)*, Fortaleza, Ceará, Brazil. ACM Press.

Rowstron, A. et P. Drusche (2001). Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*.

Stoica, I., R. Morris, D. Karger, M. F. Kaashoek, et H. Balakrishnan (2001). Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM Conference*, San Diego.

Sykes, D., W. Heaven, J. Magee, et J. Kramer (2008). From goals to components : a combined approach to self-management. In *SEAMS '08 : Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pp. 1–8. ACM.

Summary

Adaptive applications modify their behavior according to the current execution context by changing from a consistent variant to another one. The variants include implementations, configurations, parameters, architectures, etc. In the context of distributed, component-base applications, specifying consistent variants and transitions between them is critical to ensure the correctness of the collaborations after adaptation. In this paper, we present an approach that helps developpers build correct architectural variants and transitions between them for such class of applications