Déploiement sensible au contexte et reconfiguration des applications dans les sessions collaboratives

Emir Hammami

T H E S E

de

DOCTORAT de l'UNIVERSITE de TOULOUSE

Délivré par l'Université Toulouse III – Paul Sabatier

Ecole Doctorale Systèmes

Discipline : Systèmes Informatiques

présentée et soutenue

par

Emir HAMMAMI

Le 6 décembre 2007

Titre : Déploiement sensible au contexte et reconfiguration des applications dans les sessions collaboratives

_____

Directeur de thèse : Thierry VILLEMUR

Co Directeur de thèse : Michel DIAZ

_____

JURY

M. Nazim AGOULMINE, Rapporteur
M. Hervé GUYENNET, Rapporteur
M. Guy JUANOLE, Examinateur Président du jury
M. Karim DJOUANI, Examinateur
M. François VERNADAT, Examinateur
M. Thierry VILLEMUR, Directeur de thèse

RESUME

Le déploiement sensible au contexte et la reconfiguration des applications dans les sessions collaboratives sont les processus permettant de gérer la distribution initiale et ultérieure des outils collaboratifs sur les nœuds des participants en prenant en compte diverses contraintes. Ces contraintes proviennent de la structure de la session, des environnements d'exécution et des relations entre les participants.

La majorité des travaux liés au déploiement et à la reconfiguration proposent des solutions qui couvrent essentiellement les deux premiers points mais qui n'accordent pas beaucoup d'importance au dernier point. Ainsi, les applications déployées couvrent les besoins des utilisateurs et sont compatibles avec le contexte local mais l'interopérabilité avec les applications déjà déployées sur les nœuds des voisins n'est pas vérifiée. De plus, le contrôle du déploiement et la découverte des applications requises se fait de façon centralisée et nécessite l'intervention humaine.

Nous proposons une approche de déploiement et de reconfiguration automatique basée sur un algorithme décentralisé qui s'exécute sur chaque nœud de déploiement. Cet algorithme utilise des modèles abstraits pour générer des configurations de déploiement valides permettant de respecter les diverses contraintes. Nous avons développé une plate-forme Pair-à-Pair offrant des modules génériques pour supporter cet algorithme. Enfin, nous avons réalisé des prototypes et nous avons mené des tests de performance afin d'évaluer expérimentalement notre approche.

MOTS-CLES

Déploiement sensible au contexte, reconfiguration, session collaborative, pair-à-pair, JXTA

Remerciements

Mes remerciements s'adressent tout d'abord à mes parents, ma femme ainsi que sa famille, mes frères et mes sœurs et tous ceux qui m'ont aidé à travailler dans cette thèse.

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS), dirigé durant cette période par Messieurs M. GHALLAB et R. CHATILA. Je tiens à les remercier pour m'avoir accueilli en tant que doctorant dans ce laboratoire.

Je remercie l'ensemble des responsables successifs du groupe Outils Logiciels pour la Communication (OLC), Messieurs J.P. COURTIAT et F. VERNADAT, qui, par leur animation scientifique, contribuent à maintenir un cadre de recherche performant.

Je remercie également mes directeurs de thèse Michel DIAZ et Thierry VILLEMUR pour leurs conseils, pour la lecture attentive de ce manuscrit et pour leur aide dans les publications.

Je suis très reconnaissant à :

M. Nazim AGOULMINE, Professeur à l'Université d'Evry Val d'Essonne,
M. Hervé GUYENNET, Professeur à l'Université de Franche-Comté,
M. Guy JUANOLE, Professeur à l'Université de Toulouse,
M. Karim DJOUANI, Professeur à l'Université Paris 12,
M. François VERNADAT, Professeur à l'Institut National des Sciences Appliquées de Toulouse
M. Thierry VILLEMUR, Maître de Conférences à l'Université Toulouse II,

pour l'honneur qu'ils me font en participant à ce jury de thèse, et plus particulièrement à Messieurs AGOULMINE et GUYENNET qui ont accepté la charge de Rapporteur.

J'adresse un grand merci à tous mes collègues du groupe Outils Logiciels pour la Communication que j'ai côtoyés durant ces années. De façon plus précise, je remercie Khalil DRIRA ainsi que sa famille pour leur accueil durant mon séjour en France.

Plan

Figures

Tableaux

Chapitre 1 Introduction

L'ingénierie coopérative distribuée est fortement d'actualité aujourd'hui, car elle constitue un moyen pour améliorer la qualité et la productivité dans les entreprises. Etre capable de travailler à distance de façon coopérative est un rêve que partagent les professionnels de nombreux domaines d'activité. De plus, la généralisation des moyens de communication et de traitement de l'information et leur déploiement sans cesse croissant, encouragent la recherche dans ces domaines, ceci afin d'anticiper les besoins des utilisateurs et de leur offrir une réorganisation adaptée aux futures activités d'entreprise.

1.1 Motivation et problématique

Lorsqu'un participant se connecte à une session de travail collaboratif, il est invité à travailler avec les autres membres à travers des outils collaboratifs. Ces outils permettent l'échange (en production et/ou en consommation) de divers types de données comme la vidéo, l'audio ou simplement le texte. Une phase préalable de déploiement est alors nécessaire. Cette phase vise à fournir au nouveau membre les applications requises pour le bon déroulement de son activité. De plus, la structure de la session peut évoluer dans le temps entraînant le changement des rôles des participants. Ce qui implique l'utilisation d'autres outils collaboratifs. Une phase de reconfiguration est ainsi nécessaire pour garantir que les applications soient conformes au nouvel état.

Dans les approches classiques, les environnements de travail collaboratif présupposent la disponibilité des outils collaboratifs à activer chez chacun des participants, avant tout démarrage de la session. Ces approches adoptent donc un déploiement caractérisé par les aspects suivant :

- **Déploiement manuel.** L'administrateur ou le participant effectue lui-même les installations des applications. Bien que cette solution ne nécessite pas des outils spécialisés dans le déploiement, elle requiert des connaissances élémentaires des procédures d'installation de la part des participants, ce qui n'est pas toujours possible. De plus, il est difficile de choisir manuellement les applications appropriées pour qu'elles puissent s'exécuter correctement en présence d'un environnement d'exécution spécifique. Une autre difficulté réside dans la vérification de l'interopérabilité avec les applications déployées par les autres nœuds de la session. Gérer manuellement les installations devient rapidement une tâche difficile voire impossible dans certains cas et produit une perte de temps considérable.

- **Déploiement exhaustif**. Généralement, on déploie toutes les applications, celles qui sont utilisées pour une session spécifique et celles qui ne le sont pas. Certes, cette solution permet de ne pas effectuer un autre déploiement lorsque l'on passe d'un schéma de collaboration vers un autre. Mais, en contrepartie, elle nécessite plus de temps lors du processus de déploiement et plus d'espace disque, espace qui peut être difficile à se procurer pour des équipements nomades à ressources limitées.

- **Déploiement statique**. Le déploiement est effectué selon des critères prédéfinis et inchangés tout au long des phases de collaboration. Cette solution s'applique plus aux sessions figées qui n'évoluent pas dans le temps, cette solution est parfaite. Cependant, les nouveaux environnements de travail collaboratif autorisent les sessions dynamiques dont la structure (ou schéma de collaboration) peut évoluer dans le temps. Les applications requises pour le nouvel état ne sont pas nécessairement déployées à l'avance sur tous les nœuds.

Avec le développement de l'informatique, beaucoup de progrès ont été réalisés tant au niveau de l'infrastructure matérielle que logicielle. Ainsi, de nouvelles exigences apparaissent :

- **Plus de choix concernant les outils collaboratifs**. Le domaine du génie logiciel émerge rendant le développement des outils collaboratifs plus facile grâce aux technologies de la réutilisation et des composants. Le choix entre ces outils devient de plus en plus difficile manuellement.

- **Contextes variés**. Les nœuds de déploiement offrent des environnements d'exécution de plus en plus hétérogènes allant des ordinateurs de bureau vers les assistants personnels et les téléphones portables. Les systèmes d'exploitation sont variés (Windows, Linux, Novell, …). Il est nécessaire de prendre en compte les contraintes logicielles et matérielles de chaque nœud de déploiement.

- **Evolution de la session**. L'activité collaborative en cours peut introduire des changements dynamiques dans la session. Ces changements interviennent particulièrement au niveau de l'espace des participants [1] : (i) Entrées/sorties des participants. (ii) Changements des rôles des participants.

Partant de ces constats, un déploiement automatique, personnalisé et dynamique est alors nécessaire pour identifier et déployer les applications les plus aptes à répondre aux besoins de chaque participant tout en respectant la structure globale de la session.

Du point de vue architectural, la majorité des prototypes de déploiement tels que Planit [2], CADeComp [3], Software Dock [4], et ORYA [5], repose sur une conception centralisée. De façon schématique, le système est composé d'un ensemble de nœuds de déploiement, d'un ensemble de fournisseurs d'applications et d'un gestionnaire de déploiement (figure 1.1). Les nœuds de déploiement sont les hôtes sur lesquels les applications vont être installées. Les fournisseurs hébergent et mettent à disposition les applications pour les autres nœuds. Le gestionnaire de déploiement se charge de faire la liaison entre les nœuds de déploiement et les fournisseurs. Il est aussi responsable des opérations d'adaptation pour fournir à un nœud les applications les plus aptes à répondre à ses besoins. Une telle approche en mode client-serveur, appelée architecture centralisée, est représentée schématiquement par la figure 1.1.a.

L'apparition du Pair-à-Pair a contribué fortement à revoir le déploiement en adoptant une nouvelle forme de distribution des applications. Les nœuds de déploiement jouent désormais un double rôle, celui d'hôte pour accueillir les applications, mais aussi celui de fournisseur d'applications. En effet, les applications rapatriées sur un nœud sont aussi mises à la disposition des autres nœuds. Cette approche bénéficie des avantages apportés par le modèle Pair-à-Pair (équilibrage de charge, tolérance aux pannes, plus de choix parmi des

applications,…). Cependant, dans cette architecture hybride, la gestion du déploiement est toujours effectuée par un ou plusieurs contrôleurs centralisés. La plate-forme Hydra [6] appartient à cette catégorie. La figure 1.1.b décrit schématiquement cette approche.

De nos jours, il est possible de mettre en place des sessions collaboratives sans recourir à une administration centralisée. Adopter une architecture de déploiement totalement décentralisée, non seulement au niveau de la répartition des applications mais aussi au niveau du contrôle du déploiement, devient alors obligatoire. Les nœuds ne sont plus attachés à une entité centrale et gardent ainsi leur caractère autonome. Le travail décrit dans [7], et la plate-forme CoDeWAN [8] appartiennent à cette catégorie. Une telle architecture décentralisée, est représentée par la figure 1.1.c.

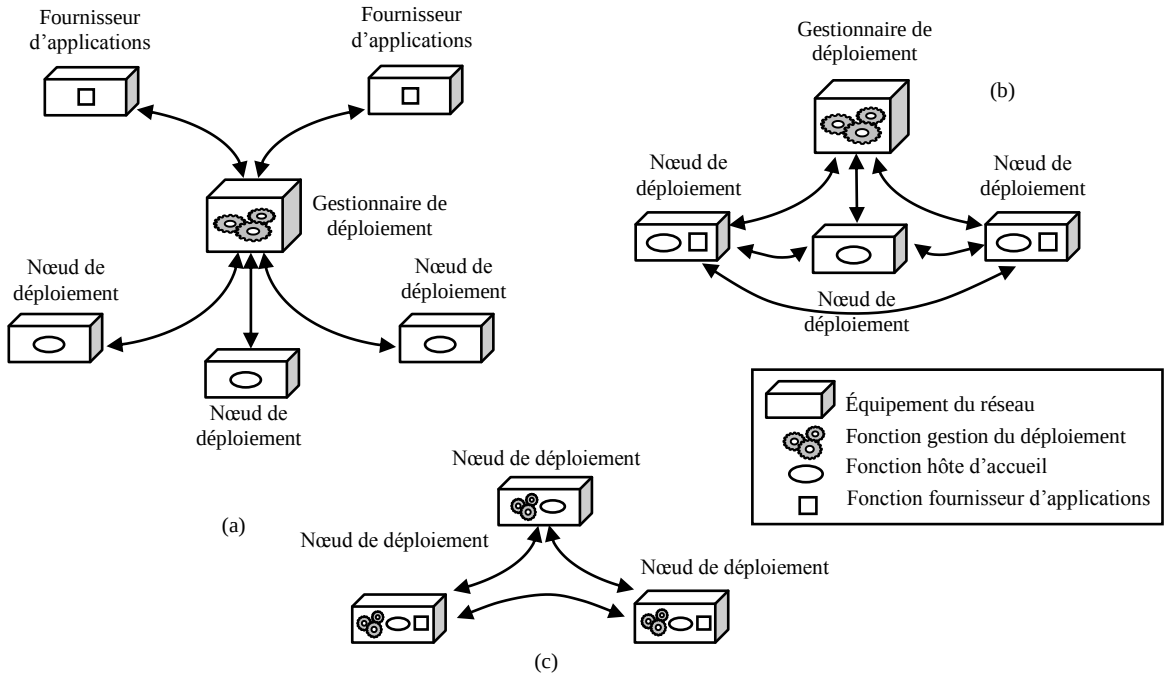


Figure 1.1. Architectures de déploiement

Le tableau 1.1 synthétise les caractéristiques des trois architectures de déploiement que nous avons identifiées.

Type d'architecture	**Centralisée (a)**	**Hybride (b)**	**Décentralisée (c)**
Gestion du deployment	Centralisée	Centralisée	Décentralisée
Distribution des applications	Client-Serveur	Pair-à-Pair	Pair-à-Pair

Tableau 1.1. Déploiement et distribution des applications

1.2 Position de notre travail et contributions

Dans cette thèse, nous étudions les divers aspects liés à la problématique du déploiement et de la reconfiguration. Nous proposons ensuite un algorithme de déploiement sensible au contexte (local et global). Cet algorithme s'exécute sur chaque nœud. L'objectif est de permettre à un participant voulant accéder à la session courante d'avoir automatiquement et de façon transparente les applications qui lui font défaut, tout en respectant les contraintes issues de la compatibilité avec le contexte local et d'interopérabilité avec les applications situées sur les autres nœuds voisins. Cet algorithme

supporte aussi la reconfiguration (c'est-à-dire le déploiement ultérieur) lorsque la session évolue dans le temps. Notre travail utilise plus particulièrement des sessions structurées, où un schéma de collaboration prédéfini gouverne les relations entre participants.

Nous avons conçu et réalisé une plate-forme Pair-à-Pair composée de modules génériques qui supportent cet algorithme. Ces modules permettent tout d'abord de décrire les informations requises pour le déploiement : le contexte local, global et les caractéristiques des applications. Ils permettent ensuite de publier et de rapatrier ces applications, après les avoir cherchées dans le réseau P2P. Ainsi notre travail se place dans la catégorie des architectures décentralisées (du tableau 1.1).

Les principales contributions de cette thèse sont les suivantes :

- Proposer des modèles de sessions, d'applications et de nœuds de déploiement permettant d'abstraire les informations nécessaires à l'automatisation du processus du déploiement et de la reconfiguration;
- Proposer un algorithme de déploiement et de reconfiguration qui supporte les opérations de déploiement initial et ultérieur suite à l'évolution de la session ;
- Définir une architecture en couches implantant l'algorithme et qui utilise les modèles proposés. Cette architecture modulaire offre un ensemble d'APIs génériques pour construire des applicatifs plus complexes.
- Implanter cette architecture en utilisant le langage Java et la plate-forme JXTA spécialisée dans les applications Pair-à-Pair ;
- Développer plusieurs prototypes pour, au travers d'expérimentations, évaluer l'approche proposée.

1.3 Organisation de la thèse

Le présent document est organisé comme suit :

Le chapitre 2 fournit un état de l'art du déploiement. Il se divise en trois grandes parties. La première donne les définitions des termes et des concepts liés au thème du déploiement. La deuxième partie est dédiée aux systèmes de déploiement insensibles au contexte. La troisième partie se focalise sur les systèmes de déploiement sensibles au contexte.

Le chapitre 3 donne la spécification de l'algorithme de déploiement. Trois parties sont présentées. La première identifie les phases du déploiement et de la reconfiguration. La deuxième partie décrit les modèles proposés pour automatiser le processus du déploiement. Enfin, la troisième partie détaille l'algorithme de déploiement et de reconfiguration.

Le chapitre 4 est dédié à la plate-forme de déploiement qui supporte l'algorithme proposé. Il introduit tout d'abord, les fonctionnalités requises pour le déploiement et la reconfiguration. Ensuite, l'architecture en couches de notre proposition sera présentée en détaillant les modules offerts par chaque couche.

Le chapitre 5 s'intéresse de plus près à l'étude expérimentale de notre API de déploiement et de reconfiguration. Nous commençons par montrer les réalisations qui ont été faites conformément à l'architecture proposée. Cinq prototypes ont été développés à cet effet. Ensuite, nous évaluons l'algorithme de déploiement à travers des tests de simulations. La dernière partie de ce chapitre évalue par des tests réels, les modules permettant d'effectuer les opérations de découverte et de rapatriement entre les nœuds d'un réseau Pair-à-Pair.

Nous concluons la thèse dans une dernière partie qui présente le bilan du travail réalisé et les contributions apportées. Nous listons également les perspectives de continuation de notre travail.

Chapitre 2 Etat de l’art et synthèse

2.1 Introduction

Depuis le début de l'informatique, les utilisateurs avaient besoin d'installer des applications sur leurs ordinateurs. Le premier mode de déploiement se basait sur l'utilisation de la disquette pour copier un programme exécutable sur un autre ordinateur. Encouragé par les progrès réalisés tant au niveau de l’infrastructure réseau qui offre désormais des débits plus confortables, qu’au niveau des machines elles-mêmes (vitesse du processeur, taille de la mémoire et espace disque), plusieurs modes de déploiement ont apparus.

Dans ce chapitre état de l'art, nous introduisons d'abord les principaux termes et définitions qui représentent les concepts fondamentaux énoncés dans ce mémoire. Nous mettons ensuite l'accent sur les systèmes de déploiement. Ainsi, nous décrivons en premier lieu des solutions de déploiement qui ne tiennent pas compte du contexte. Puis, nous étudions des solutions de déploiement sensible au contexte. Chaque partie se divise en deux catégories : déploiement sur un seul nœud et déploiement sur plusieurs nœuds. Enfin, nous présentons les principaux modèles et langages descriptifs utilisés pour automatiser le processus de déploiement et de reconfiguration.

2.2 Définitions

Cette section décrit l'étude que nous avons menée pour clarifier les concepts liés au déploiement et à la reconfiguration d'application. Cette étude est valable aussi bien pour le déploiement des applications dans les sessions collaboratives que pour le déploiement des composants des applications distribuées dans un réseau de nœuds. Puisque le thème de ce mémoire concerne le déploiement dans les sessions collaboratives, il nous semble judicieux d'introduire par la même occasion quelques concepts relatifs au travail collaboratif.

2.2.1 Session collaborative

Une session est définie comme une communication multipartie entre différents participants. Ces derniers travaillent sur des tâches individuelles et partagées. Plusieurs plates-formes de travail collaboratif supportent ces deux activités. Les participants se connectent depuis des équipements informatiques tels que les ordinateurs de bureau ou plus récemment les téléphones mobiles. Une session peut être synchrone ou asynchrone :

- **Synchrone.** Dans une session synchrone, les participants sont connectés en même temps. Ils utilisent des applications logicielles afin d’aboutir au but escompté par leur collaboration. Un exemple de ce type de session est une téléconférence pour superviser une opération médicale.

- **Asynchrone**. Par contre, dans une session asynchrone, les utilisateurs ne communiquent qu'au travers de média asynchrones tel que le courrier électronique. La co-présence des différents membres n'est pas nécessaire au déroulement de la session.

Nous avons identifié à l'intérieur des sessions collaboratives trois espaces : l'espace utilisateur, l'espace outil collaboratif et l'espace donnée. L'activité collaborative en cours peut introduire des changements dynamiques dans la session. Ces changements interviennent au niveau des trois espaces identifiés [1] :

- Au niveau de l'espace utilisateur, deux changements sont possibles : (i) **Entrées/sorties d'utilisateurs**. Les utilisateurs ont la possibilité de joindre ou de quitter la session à n'importe quel instant. (ii) **Changements des rôles utilisateurs**. Un rôle opérationnel est assigné à chaque utilisateur durant la phase active de la session. Ce rôle peut évoluer dans le temps.

- Plusieurs actions sont associées aux outils collaboratifs. Ces actions modifient l'état des outils (activé/désactivé) et changent leur répartition en fonction de l'évolution en ligne de la session.

- Au niveau de l'espace donnée, les actions et les modifications possibles concernent la création, la suppression et la mise à jour des flux de données.

2.2.2 Stratégies de déploiement

Nous distinguons deux stratégies de déploiement : *push et pull*.

- ***Push***. L'initiative du déploiement est donnée à l'administrateur ou à un nœud superviseur. Les applications à déployer seront transférées au(x) nœud(s) de déploiement sans qu'il y ait une demande au préalable. Généralement, le nœud superviseur contient un programme principal offrant une interface à travers laquelle l'administrateur sélectionne les applications à déployer et les nœuds de déploiement. Du côté du nœud du déploiement, un programme accueille les applications transférées.

- ***Pull***. A l'opposé de la stratégie *push*, le nœud du déploiement initialise le processus de déploiement. Généralement, l'utilisateur exprime une requête de déploiement. S'il y a une réponse favorable, l'application recherchée sera déployée sur son nœud.

2.2.3 Catégories de déploiement

Nous recensons deux catégories de déploiement : statique et dynamique, appelées aussi respectivement, explicite et implicite [2].

- **Statique (explicite)**. La correspondance de l'application (ou du composant) avec le nœud de déploiement est définie de façon explicite par l'administrateur. Cette correspondance se réalise avant le début du processus du déploiement.

- **Dynamique (implicite)**. Le choix de l'application (ou du composant) et son attribution au nœud de déploiement se fait de manière intelligente et automatique pendant le processus de déploiement.

2.2.4 Types de déploiement

Il existe deux types de déploiement : initial et ultérieur.

- **Initial.** Le déploiement initial fait référence au premier déploiement des applications à utiliser. Dans le cas des applications distribuées orientées composants, il s'agit du premier déploiement des composants sur les nœuds.

- **Ultérieur**. Appelé aussi reconfiguration. Son but est de permettre à un système d'évoluer pendant son exécution. Dans le cas des applications distribuées orientées composants, la reconfiguration s'effectue pendant l'exécution de l'application sans l'arrêter, suite aux changements qui peuvent intervenir dans son environnement. Dans le cas d'une session collaborative, la reconfiguration s'impose quand les rôles des participants évoluent.

2.2.5 Contrôle du déploiement

Le contrôle du déploiement peut être centralisé ou décentralisé.

- **Centralisé**. Dans ce cas, une entité principale dirige les opérations de déploiement. Les nœuds de déploiement ne sont pas autonomes et dépendent fortement de cette entité. L'avantage de ce schéma est sa facilité de mise en œuvre. Cependant, il n'est pas approprié pour les réseaux de type ad-hoc.

- **Décentralisé**. Il n'existe pas d'entité centralisée pour diriger le déploiement. Chaque nœud contient un gestionnaire de déploiement qui est en interaction avec les nœuds voisins. Ce schéma s'accorde parfaitement avec les besoins des nouvelles applications de l'informatique ambiante. L'inconvénient majeur résulte de sa difficulté de mise en œuvre.

2.2.6 Performance du déploiement

Les deux principaux critères permettant d'évaluer une approche de déploiement sont : la cohérence et le degré d'automatisation.

Cohérence

On distingue la cohérence locale et la cohérence globale. Dans le premier cas, elle concerne seulement le nœud de déploiement indépendamment des autres nœuds. Dans le deuxième cas, elle concerne tous les nœuds impliqués dans la session. En s'inspirant des travaux sur le déploiement des applications distribuées à base de composants [9], la cohérence locale peut être caractérisée par les points suivants :

- **Sécurité**. Une opération de déploiement ou de reconfiguration mal faite ne doit pas conduire le nœud de déploiement dans un état critique.

- **Complétude**. Le déploiement et la reconfiguration doivent se terminer au bout d'un certain temps. Ceci est utile pour ne pas bloquer le nœud indéfiniment.

- **Terminaison correcte (C*orrectness*).** Cette propriété est assurée lorsque le déploiement se termine de façon correcte par rapport aux objectifs fixés au début. On parle aussi de *terminaison correcte totale* quand il s'agit de satisfaire tous les buts du déploiement, et de *terminaison correcte partielle* quand il s'agit de répondre seulement à une partie des buts fixés.

- **Retour en arrière (*Rollback*)**. Cette propriété permet à un nœud de regagner son état avant le déploiement ou la reconfiguration. Le retour en arrière est utile pour faire face aux erreurs qui peuvent se passer lors du déploiement.

- **Passage à l'échelle (*Scalability*).** Cette propriété signifie que les performances du déploiement et de la reconfiguration ne se dégradent pas, ou se dégradent légèrement, lorsque les besoins du nœud de déploiement et le nombre de nœuds voisins augmentent.

- **Activation.** La propriété d'activation implique que le déploiement et la reconfiguration doivent réellement s'exécuter quand la situation l'exige.

- **Rapidité.** Le déploiement ou la reconfiguration des applications sur un nœud doivent se faire rapidement, sans compromettre le fonctionnement du système.

- **Choix du moment (*Well-timedness*)**. Bien choisir le moment pour effectuer les opérations du déploiement et de reconfiguration est important pour arriver à la cohérence locale.

Pour la cohérence globale, on retrouve les mêmes points caractérisant la cohérence locale, hormis la propriété de retour en arrière :

- **Sécurité**. L'échec du déploiement sur un nœud ne doit pas perturber le fonctionnement de la session. Les autres nœuds continuent leurs activités normalement.

- **Complétude**. Cette propriété concerne seulement la reconfiguration puisqu'elle implique tous les nœuds de la session. Ainsi, il faut que l'ensemble de ces nœuds finisse les modifications nécessaires au bout d'un certain temps. La complétude globale dépend donc de la complétude locale.

- **Terminaison correcte (*Correctness*).** Il faut que tous les nœuds concernés par la reconfiguration réussissent le déploiement selon les objectifs fixés à l'avance. On retrouve la notion de *terminaison correcte totale* quand tous les nœuds répondent aux objectifs, et de *terminaison correcte partielle* quand seulement une partie des nœuds ont réussi la reconfiguration.

- **Passage à l'échelle (*Scalability*).** Les performances du déploiement et de la reconfiguration ne se dégradent pas lorsque le nombre des nœuds de la session évolue.

- **Activation.** Dans le cas de la cohérence globale, la propriété d'activation concerne seulement la reconfiguration. Elle signifie que la reconfiguration doit réellement s'exécuter quand la session évolue et que cette évolution nécessite une reconfiguration.

- **Rapidité.** L'évolution de la session par l'ajout ou le retrait d'un nœud ou par le changement de rôle d'un utilisateur, doit être traitée dans un temps raisonnable pour maintenir une certaine qualité dans l'exécution de l'activité collaborative.

- **Choix du moment (*Well-timedness*)**. La cohérence globale dépend de la cohérence locale de tous les nœuds de déploiement. La session est cohérente quand tous les nœuds sont cohérents. Ainsi, le choix du moment pour effectuer le déploiement et la reconfiguration au niveau de chaque nœud est important pour arriver à la cohérence globale.

Degré d'automatisation

Le degré d'automatisation représente la capacité de l'algorithme de déploiement à réaliser de façon autonome les opérations d'adaptation et de reconfiguration. Plus l'intervention humaine est minimale, plus le degré d'automatisation est fort. Les systèmes actuels tendent plus vers des systèmes autonomes dans lesquels le degré d'automatisation joue un rôle très important [10], [11]. Comme nous avons vu dans la section 2.3.2 concernant le déploiement sur plusieurs nœuds, le niveau d'abstraction renforce aussi l'automatisation du déploiement. Ainsi, les points suivants permettent de caractériser le degré d'automatisation :

- L'instant du déclenchement du déploiement ou de la reconfiguration ;

- La méthode employée pour connaître les besoins des utilisations ;

- La procédure utilisée pour vérifier la compatibilité des applications par rapport au contexte d'exécution.

2.2.7 Sensibilité au contexte

La notion de contexte regroupe l'ensemble des paramètres qui entourent une entité. Par exemple, dans le cas du processus de déploiement, l'entité est l'application. Les caractéristiques de l'environnement d'exécution forment le contexte local de l'application. Ces informations influent sur le choix des applications ou des composants à déployer. La prise en compte de ces informations dans le déploiement et la reconfiguration, se fait de deux manières [12], [13] :

- **Mode passif**. Dans ce cas, une tierce entité effectue les opérations d'adaptabilité au contexte. Typiquement c'est le gestionnaire de déploiement qui s'occupe de récolter les informations de contexte et de décider des opérations à faire.

- **Mode actif.** L'application elle-même est capable de percevoir et d'analyser son contexte, et de s'y adapter. Les modules permettant l'adaptabilité sont intégrés dans l'application qui renferme donc une partie fonctionnelle et une partie contrôle du déploiement.

Dans ce qui suit, nous apportons une classification des différents cas de compatibilité des ressources nécessaires à une application par rapport aux ressources disponibles sur un nœud de déploiement. Pour cela, on distingue entre divers granularités : fine, moyenne et forte.

Le tableau 2.1 résume les différents cas de compatibilité des ressources nécessaires à une application par rapport aux ressources disponibles sur un nœud de déploiement.

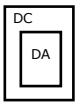
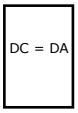
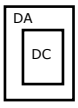
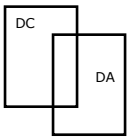
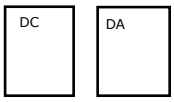
Compatible		**Non compatible**		
		Partiellement compatible		**Incompatible**
Sur-ensemble	**Exact**	**Sous-ensemble**	**Intersection**	**Disjoint**
				
$DA \subset DC$		$DC \subset DA$	$DC \cap DA \neq \phi$	$DC \cap DA = \phi$

Tableau 2.1. Degré de compatibilité

Avec :

- DA : descripteur d'application,
- DC : descripteur du contexte d'exécution.

Granularité fine

Cette classification s'applique plus sur la comparaison entre le descripteur d'application et le descripteur du contexte d'exécution [14].

- **Sur-ensemble**. Le descripteur du contexte d'exécution est un sur-ensemble du descripteur d'application. Dans ce cas, le nœud de déploiement satisfait tous les besoins de l'application, et en plus il offre d'autres ressources. L'ensemble des ressources requises non satisfaites est ainsi l'ensemble vide.

- **Exact**. Le descripteur du contexte d'exécution a les mêmes attributs que le descripteur d'application. L'ensemble des ressources requises non satisfaites est l'ensemble vide.

- **Sous-ensemble.** Le descripteur du contexte d'exécution est un sous-ensemble du descripteur d'application. Le nœud de déploiement ne permet pas de satisfaire toutes les exigences de l'application, ce qui risque de compromettre son bon fonctionnement. Le nombre de ressources requises non satisfaites est supérieur à zéro.

- **Intersection**. Le descripteur du contexte d'exécution et le descripteur d'application ont des attributs communs. Le nœud de déploiement satisfait seulement une partie des exigences de l'application, ce qui risque de compromettre son bon fonctionnement. Le nombre de ressources requises non satisfaites est aussi supérieur à zéro.

- **Disjoint**. Le descripteur du contexte d'exécution et le descripteur d'application ne partagent aucun attribut. Le nombre des ressources requises non satisfaites est égal au nombre des ressources requises.

Les résultats de la comparaison Sur-ensemble et Exact signifient que l'application est compatible au contexte d'exécution. Les résultats Sous-ensemble, Intersection et Disjoint signifient que l'application n'est pas compatible au contexte d'exécution.

Granularité moyenne

La granularité moyenne ne considère que trois niveaux :

- **Complètement compatible**. Toutes les ressources requises par l'application sont satisfaites par les ressources disponibles sur le nœud de déploiement. Dans ce cas, l'application peut s'exécuter correctement.

- **Partiellement compatible**. Seulement une partie des ressources requises par l'application sont satisfaites par les ressources disponibles sur le nœud de déploiement. On n'est donc pas sûr du bon fonctionnement de l'application.

- **Incompatible**. Les ressources disponibles sur le nœud de déploiement ne couvrent pas les besoins de l'application. L'application ne peut pas s'exécuter correctement.

Granularité forte

Pour la granularité forte, on regroupe les éléments partiellement compatibles et incompatibles en un ensemble non compatible. Une application candidate au déploiement est soit compatible, soit non compatible.

Variation de contexte

Le contexte d'exécution est amené à évoluer dans le temps. Les ressources disponibles peuvent prendre de nouvelles valeurs qui risquent de ne plus répondre aux exigences de l'application, ce qui peut causer son disfonctionnement. Ces changements doivent être notifiés et pris en compte lors de la reconfiguration.

2.2.8 Catégorie de reconfiguration

La reconfiguration, du fait de sa nature dynamique, est une opération plus complexe à mettre en œuvre que le simple déploiement initial car elle tient compte de plusieurs facteurs dynamiques comme la variation de la structure de la session. A partir des travaux de [9] sur les applications à base de composants, nous énumérons les catégories de reconfiguration dans le cas des sessions collaboratives :

- **Correctionnelle**. Après avoir réalisé le déploiement, il est possible que certaines applications ne fournissent pas les services attendus ou que les implantations déployées ne fonctionnent pas correctement. La solution consiste à refaire le déploiement pour remplacer les applications « défectueuses » par de nouvelles versions. Ces versions fournissent les mêmes fonctionnalités. Elles se contentent simplement de corriger les défauts.

- **Adaptative**. L'environnement d'exécution peut être amené à évoluer, ce qui peut perturber le fonctionnement des applications déployées. Par exemple, les liens de communications peuvent être saturés, ce qui peut engendrer des difficultés pour utiliser certaines applications gourmandes en ressources réseau, comme les visioconférences. On décide alors de communiquer simplement par des applications de dialogues textuels, plus économes.

- **Evolutive**. Les besoins des utilisateurs peuvent évoluer. La reconfiguration évolutive permet de couvrir ces nouveaux besoins, soit en ajoutant d'autres applications tout en gardant les anciennes, soit en remplaçant les anciennes par de nouvelles applications.

- **Perfective**. L'objectif de ce type de reconfiguration est d'améliorer le cadre du travail du participant en lui fournissant par exemple des applications plus performantes que celles déjà déployées. La reconfiguration perfective devient aussi nécessaire voire obligatoire dans certains cas. Par exemple, pour éviter la dégradation d'une application qui répond à plusieurs utilisateurs, on peut modifier ces paramètres pour qu'elle n'accepte qu'un nombre limité de connexions.

2.3 Systèmes de déploiement

Plusieurs travaux traitent des divers aspects du déploiement. Nous avons classé l'étude de ces travaux en deux groupes de travaux liés au déploiement :

- Indépendamment du contexte d'exécution ;
- Sensible au contexte.

2.3.1 Déploiement automatique indépendant du contexte

Les travaux liés au déploiement automatique indépendamment du contexte d'exécution prennent deux directions selon que l'environnement cible concerne un seul hôte ou bien plusieurs hôtes :

Déploiement sur un seul nœud

Cette activité de déploiement a pour objectif d'installer une application sur un nœud bien précis. Nous décrivons dans la suite, les principaux outils de déploiement offrant cette possibilité.

Le premier outil est Java Web Start (JWS) [15] développé par Sun Microsystem. Cet outil fournit une solution de déploiement des applications java, solution basée sur le protocole Java Network Launching Protocol (JNLP). Ainsi, avec JWS, il est possible de télécharger sur le disque local la dernière version de l'application Java disponible sur un serveur Web. L'accès se fait soit par un navigateur web pointant sur le document JNLP décrivant l'application, soit par le gestionnaire de JWS installé localement.

InstallShield [16] et Microsoft Windows Installer [17] permettent l'installation de logiciels en se basant sur des scripts d'installation pour recopier les fichiers requis et enregistrer les bibliothèques nécessaires. Ils sont destinés uniquement à des systèmes d'exploitation de type Windows.

D'autres outils de déploiement intégrés dans les systèmes d'exploitation de type Unix existent. Nous citons particulièrement les systèmes de paquetage Linux RedHat RPM [18], les commandes HP-UX [19], ainsi que les commandes *pkg* disponibles dans Sun Solaris. Ces outils offrent particulièrement des fonctionnalités de gestion de paquetage comme la création d'un paquetage, l'installation, la vérification de l'intégrité du système installé et, la désinstallation [20].

Parmi les travaux académiques, nous retrouvons celui présenté dans [7], [21]. L'auteur propose une plate-forme logicielle permettant à un nœud connecté à un réseau Pair-à-Pair, de rechercher et d'installer un composant OSGi appelé *bundle*. L'auteur caractérise une plate-forme de déploiement par quatre concepts : (1) l'archive de transport qui constitue le format de compression, (2) le descripteur de déploiement décrivant l'archive de transport, (3) la dépendance de composants exprimant les liens entre composants et ainsi contrôlant si un paquetage peut être déployé ou non, (4) le cycle de vie du composant (installé, démarré, arrêté, supprimé).

La plate-forme CODEWAN (COmponent DEployment in Wireless Ad hoc Networks) [8] s'adresse à la problématique du déploiement des applications pour les réseaux ad hoc. Un réseau Ad hoc est un réseau qui apparaît et évolue d'une manière spontanée suivant les

appareils participant dans le réseau. Chaque équipement mobile maintient un dépositaire local qui peut stocker un ensemble de composants logiciels. De cette façon, les équipements voisins peuvent s'échanger leurs propres composants logiciels en se basant sur des interactions en mode Pair-à-Pair. Une fois que l'utilisateur exprime une requête de déploiement, le gestionnaire de déploiement disponible sur son équipement procède à la collecte des composants qui lui manquent.

Déploiement sur plusieurs nœuds

L'activité de déploiement sur plusieurs nœuds a pour objectif d'installer une application sur plusieurs nœuds interconnectés. Ce processus peut se faire manuellement, mais il est source d'erreurs dans ce cas. De plus, la réalisation matérielle entraîne un coût d'administration élevé. En se basant sur [22], nous décrivons trois outils permettant le déploiement automatique sur plusieurs nœuds. Ces outils suivent des approches de déploiement basées sur des niveaux d'abstraction orientés script, langage et modèle. Plus le niveau d'abstraction est élevé, plus le niveau d'automatisation est élevé [22].

Nixes [23] est un outil utilisé pour installer, maintenir, contrôler, et surveiller des applications sur des nœuds d'un réseau. Il se compose de scripts en langages *bash* (*Bourne again shell*), d'un fichier de configuration et d'un dépositaire. Il peut résoudre automatiquement les dépendances entre les paquetages RPM issus des systèmes Unix. Pour les réseaux de petite taille, Nixes est facilement utilisable. Ainsi les utilisateurs créent un fichier de configuration pour chaque application et modifient des scripts pour le déploiement sur des nœuds cibles. L'inconvénient de cet outil est qu'il n'est pas adapté aux réseaux de grande taille et aux systèmes complexes.

SmartFrog (SF) [24] est une plate-forme pour la configuration, la description, le déploiement, et la gestion du cycle de vie des services. SF consiste en un langage déclaratif, des programmes s'exécutant sur les nœuds distants, des canevas exécutables écrits avec ce langage, et un modèle de composant. Le langage SF permet la personnalisation des configurations et la liaison statique et dynamique entre les composants pour les connecter au moment du déploiement. Le modèle de composant SF renforce la gestion du cycle de vie à travers cinq états : installé, initié, démarré, arrêté, et échoué. Ceci permet de redéployer les composants en cas d'échec.

Radia [25] est un outil de gestion de configuration orienté modèle. Pour chaque nœud de déploiement, l'administrateur définit un état désiré maintenu dans un dépositaire centralisé sous la forme d'un modèle. Les clients sur ces nœuds essaient d'atteindre l'état désiré qui déclenche les opérations de déploiement.

2.3.2 Déploiement automatique sensible au contexte

La prise en compte de l'environnement d'exécution est de plus en plus importante surtout avec les avancés réalisées tant au niveau des équipements informatiques comme les téléphones mobiles qu'au niveau des infrastructures réseau reliant ces équipements.

De nombreuses recherches ont été effectuées sur le déploiement automatique sensible au contexte (appelé aussi déploiement sémantique). Nous pouvons classer ces travaux selon que l'environnement cible concerne un seul hôte ou bien plusieurs hôtes.

Déploiement sur un seul nœud

Un nœud de déploiement offre des ressources matérielles et logicielles. Le déploiement sensible au contexte ou adaptatif tient compte de ces ressources ainsi que d'autres informations de contexte comme la localisation géographique pour les téléphones mobiles.

Pour les réseaux filaires, l'un des travaux remarquables dans ce domaine est Software Dock [4]. Cet outil fournit une solution qui s'adresse à l'ensemble du cycle de vie du déploiement. Il définit deux principaux composants : *release Dock* représente le fournisseur d'application, et *field Dock* représente le consommateur d'application c'est-à-dire le site du client. Le premier composant se comporte comme un serveur d'application. Il fournit une interface permettant à un utilisateur de choisir l'application qui l'intéresse. Chaque application est accompagnée d'agents génériques. Ces agents s'occupent de préparer la configuration de l'application sélectionnée en interprétant les informations issues de l'environnement de l'utilisateur. *Field Dock* se comporte aussi comme un serveur permettant de publier le contexte de l'utilisateur. Pour décrire la configuration de l'application à déployer, Software Dock utilise le langage Deployable Software Description (DSD) basé sur XML.

Pour les réseaux sans fil, les travaux de recherche et les prototypes industriels sont nombreux et sont regroupés sous le terme "informatique ubiquitaire". Ces travaux permettent de résoudre certains problèmes comme l'hétérogénéité des équipements, la limitation des ressources disponibles et la découverte des services. Ainsi dans [26], les auteurs proposent une infrastructure à composants pour le déploiement des services sur un équipement mobile. Un tel service est modélisé sous la forme d'un assemblage de composants logiciels distribués. Grâce à cette infrastructure, l'utilisateur peut découvrir des services adaptés à son terminal, et les déployer à la demande de façon transparente.

Le projet AMPROS [27], [28] propose un intergiciel pour le déploiement et la reconfiguration des applications distribuées orientées composants. L'architecture du service de déploiement se base sur un ensemble de composants capteurs permettant de percevoir l'environnement d'exécution et d'extraire le contexte pertinent. Ce dernier est utilisé pour choisir les composants appropriés à déployer sur le nœud de l'utilisateur. Ce travail se distingue du précédent par le fait que l'évolution du contexte pertinent influe sur les composants déployés et entraîne une activité de reconfiguration pour satisfaire le nouvel état. Un travail similaire qui s'adresse au processus de reconfiguration dans les environnements basés sur les grilles est présenté dans [29].

Déploiement sur plusieurs nœuds

Nous pouvons classer les travaux liés au déploiement sensible au contexte sur plusieurs nœuds en deux groupes. Le premier groupe se focalise sur l'installation d'une même application sur des nœuds cibles. Généralement ces nœuds appartiennent à une organisation administrée par un responsable. Le deuxième groupe traite du placement automatique des composants d'une application distribuée dans un réseau.

Open enviRonment to deploY Applications (ORYA) [5] propose une infrastructure multi-serveurs qui permet la distribution de logiciel pour un grand nombre de clients. Le contrôle du déploiement se fait de façon centralisée par un nœud gestionnaire de déploiement. Ce dernier interagit en mode client/serveur avec des nœuds serveurs d'applications pour rapatrier les applications appropriées vers les nœuds cibles selon les caractéristiques de ces derniers.

Au sujet du deuxième groupe, plusieurs solutions ont été proposées dans la littérature pour le placement des applications distribuées orientées composants. Une telle application est formée par un ensemble de composants. Chaque composant peut avoir plusieurs implantations selon l'environnement d'exécution. Les articles traitant le problème de placement ont pour but de sélectionner la bonne implantation à déployer et le bon nœud de déploiement pour satisfaire un objectif défini par avance.

Parmi ces travaux, nous citons celui présenté dans [30]. Les auteurs proposent un algorithme de placement des composants sur un réseau de nœuds. Cet algorithme s'exécute pendant le déploiement initial de l'application. L'automatisation du déploiement est possible grâce à un modèle d'application et un modèle de nœud. Le modèle d'application décrit les ressources requises pour qu'une implantation s'exécute correctement. Les auteurs font la distinction entre les ressources indispensables et les ressources facultatives représentant respectivement des contraintes fortes et des préférences.

Pour trouver la bonne configuration, deux phases sont nécessaires. La première phase consiste à sélectionner pour chaque composant de l'application, les implantations qui peuvent être déployées. Une implantation est candidate s'il existe au moins un nœud lui permettant de s'exécuter correctement. Au cours de cette première phase, l'algorithme sélectionne aussi parmi les nœuds du réseau ceux qui peuvent être utilisés pour héberger les implantations. Un nœud est candidat s'il existe au moins une implantation pouvant s'exécuter correctement. La deuxième phase consiste à affecter les implantations aux nœuds tout en maximisant la moyenne des ressources offertes par les nœuds de déploiement et le nombre des préférences satisfaites.

Un autre travail semblable au précédent, mais qui se focalise plus particulièrement sur le déploiement ultérieur (la reconfiguration), est proposé par [2]. L'auteur se base sur des techniques d'Intelligence Artificielle pour générer des configurations de déploiement en prenant en compte les contraintes temporelles et les contraintes sur les ressources disponibles. Un outil, appelé Planit, a été réalisé pour démontrer l'approche proposée. Cet outil est capable de détecter les changements du système, de générer les configurations adéquates et de publier la configuration sélectionnée.

D'autres travaux ont été consacrés au déploiement des modèles de composants. Ainsi, [31] étudie les questions et les besoins relatifs au déploiement adaptatif et automatique des applications basées sur le modèle CORBA-*LC* (*CORBA Lightweight Components*). Dans les modèles de composant traditionnels, les programmeurs décident des nœuds sur lesquels les composants vont être exécutés en se basant sur une description statique de l'architecture de l'application distribuée. Par contre, CORBA-*LC* effectue le déploiement et la gestion des dépendances automatiquement. Il offre donc la possibilité de faire le placement automatique des composants, la migration automatique des composants et l'équilibrage de charge. Ceci amène à une utilisation maximale des ressources réseau. Enfin, CORBA-*LC* s'appuie sur une architecture réflexive qui utilise des interactions en mode Pair-à-Pair, pour le déploiement des applications distribuées. Chaque nœud énonce ses caractéristiques statiques comme le type du système d'exploitation, et des caractéristiques dynamiques comme la charge du microprocesseur. Ces informations sont utilisées pour décider des implantations à déployer.

2.4 Eléments descriptifs pour automatiser le déploiement

Pour pouvoir effectuer le déploiement automatique sensible au contexte, il est assez courant de se baser sur des éléments descriptifs (modèles et/ou langages). Le descripteur d'application sert à représenter les ressources nécessaires pour le bon fonctionnement de l'application. Le descripteur de nœud sert à représenter les ressources offertes par un nœud. En se basant sur les recherches d'AbdelKarim Beloued [32], nous présentons les principaux éléments pour décrire les applications et les nœuds de déploiement.

2.4.1 Description des applications

Open Software Description (OSD)

OSD est proposé par le WWW Consortium. Il permet de décrire les paquetages logiciels dans un fichier XML en respectant une DTD bien définie. Un descripteur d'une application selon le format OSD contient trois types d'informations (figure 2.1):

```
<softpkg name="Simulator" version="1,0,0,0">
    <title>Simulator</title>
    <abstract>Simulator for testing deployment algorithm</abstract>
    <implementation>
            <os value="WinNT">
                    <osversion value="4,0,0,0"/>
            </OS>
            <processor value="P3"/>
            <language value ="en" />
            <codebase href="/implementation/simulator.cab"/>
    </implementation>
    <implementation>
            <impltype value="Java" />
            <codebase href="/implementation/simulator.jar"/>
            <dependency>
                    <codebase href="/implementation/graph.jar"/>
            </dependency>
    </implementation>
</softpkg>
```

Figure 2.1. Descripteur selon le modèle OSD

- Les informations générales sur l'application à savoir son nom, sa version, l'auteur et un texte descriptif de l'application ;

- Un certain nombre de propriétés décrivant les ressources requises par une implantation. Une application peut avoir plusieurs implantations. Chaque implantation est décrite dans un bloc à part ;

- Les informations sur les dépendances logicielles. Cette partie décrit les objets dont dépend l'application. Ces objets doivent être présents sur le nœud de déploiement.

L'avantage d'OSD est qu'il est extensible en utilisant les *namespaces* XML. Ceci permet de rajouter des ressources selon le contexte d'utilisation. Ainsi, après avoir analysé le fichier OSD, le nœud de déploiement détermine l'implantation qui s'adapte à son contexte.

L'inconvénient d'OSD est qu'il ne permet pas de définir des relations entre les ressources ainsi que des contraintes sur elles.

Deployable Software Description (DSD)

DSD est un langage utilisé dans le projet Software Dock. Il permet de décrire une application dans un fichier DSD en respectant un schéma XML bien défini. Un descripteur d'application selon DSD contient essentiellement les parties suivantes :

- Les informations générales (*Family*). Ce bloc renseigne sur le nom de l'application, le producteur, la licence, et la signature ;
- Les propriétés externes qui décrivent les caractéristiques du nœud de déploiement. Les valeurs ne sont reconnues que pendant le déploiement. DSD permet de garder la trace des valeurs de propriétés pour reconfigurer ou annuler les opérations précédentes du déploiement ;
- Les propriétés internes. Ce bloc décrit les propriétés internes de l'application comme le langage de programmation d'une implantation ;
- Les règles de composition. La définition des relations entre les propriétés se fait dans ce bloc ;
- Les artéfacts donnent le chemin des fichiers contenant les différentes configurations du logiciel ;
- Les assertions et les dépendances. Ce bloc permet de définir des contraintes sur les valeurs des propriétés internes et externes du logiciel. Une contrainte est définie par une condition qui doit être vérifiée pour que le déploiement ait lieu, au contraire d'une dépendance qui définit une procédure de résolution de conflit.

D'autres blocs existent comme ceux destinés à définir les interfaces et les services spécialisés offerts par l'application.

A l'opposé du langage OSD, l'avantage de DSD est qu'il permet de définir des relations entre les ressources ainsi que les contraintes sur elles. L'originalité de cette spécification est l'intégration des valeurs des caractéristiques du nœud dans le descripteur d'application au moment du déploiement. Cela permet d'identifier la configuration adéquate si elle existe. L'inconvénient principal est qu'il est assez difficilement compréhensible par un opérateur humain.

Java Network Launching Protocol (JNLP)

JNLP est un langage qui fait partie de l'outil Java Web Start. Pour déployer une application, il est nécessaire de la décrire par un fichier de lancement au format XML respectant le protocole JNLP. La figure 2.2 donne un exemple de descripteur JNLP.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for P2P Context-aware Deployment API-->
<jnlp
  spec="1.0+"
  codebase="http://www.laas.fr/~ehammami/archives"
  href="simulator.jnlp">
  <information>
      <title>Deployment Simulator</title>
         <vendor>LAAS - CNRS</vendor>
         <homepage href="http://www.laas.fr/~ehammami/adaptatif">
         <description>Simulator</description>
         <description kind="short">Simul</description>
         <offline-allowed/>
  </information>
  <security>
         <all-permissions/>
  </security>
  <resources>
         <j2se version="1.3+"/>
         <jar href="simul.jar" main="true" download="eager"/>
         <jar href="lib/bcprov-jdk14.jar" download="eager"/>
                     <jar href="lib/jaxen-core.jar" download="eager"/>
  </resources>
  <application-desc main-class="net.deployment.simulator.Simulator">
  </application-desc>
</jnlp>
```

Figure 2.2. Descripteur selon le modèle JNLP

L'élément <jnlp> constitue l'élément racine du document « .jnlp ». L'élément <title> permet d'indiquer le nom de l'application. L'élément <jar> montre le chemin des archives java nécessaire à l'exécution de l'application. D'autres éléments comme <description> peuvent figurer dans le document. L'avantage de JNLP est qu'il est facile à mettre en œuvre par un opérateur humain. Cependant, il permet seulement d'indiquer les archives nécessaires pour l'exécution de l'application mais pas les ressources matérielles sur le nœud de déploiement.

2.4.2 Description des ressources

Resource Descriptor Framework (RDF)

Plusieurs langages de description des ressources offertes par les nœuds de déploiement existent. Parmi eux, RDF qui est un modèle de graphe pour décrire les données et les méta-données. Il permet un traitement automatique des méta-données [33]. Un document structuré en RDF est un ensemble de triplets. Chaque triplet est une association {sujet, objet, prédicat} :

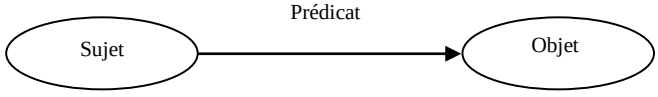


Figure 2.3. Triplet RDF

- Le sujet fait référence à la ressource à décrire par un URI (Uniform Resource Identifier) ;
- L'objet représente une autre ressource ou une valeur littérale ;
- Le prédicat définit une propriété du sujet dont la valeur est l'objet.

Un document RDF ainsi formé correspond à un graphe orienté étiqueté. Chaque triplet correspond alors à un arc orienté dont le label est le prédicat, le nœud source est le sujet et le nœud cible est l'objet. RDF est une des bases du succès du Web sémantique [33]. RDFS (RDF Schema) permet d'étendre le vocabulaire en définissant le schéma correspondant au domaine d'utilisation. Un exemple de schéma RDF est donné dans la figure 2.4. Les deux classes *Disque Dur* et *Mémoire* ont une propriété commune : *Taille*. Nous avons donc ajouté la classe *Support de mémorisation* qui a pour propriété *Taille*. Ainsi les classes *Disque Dur* et *Mémoire* seront des sous-classes de la classe *Support de mémorisation*.

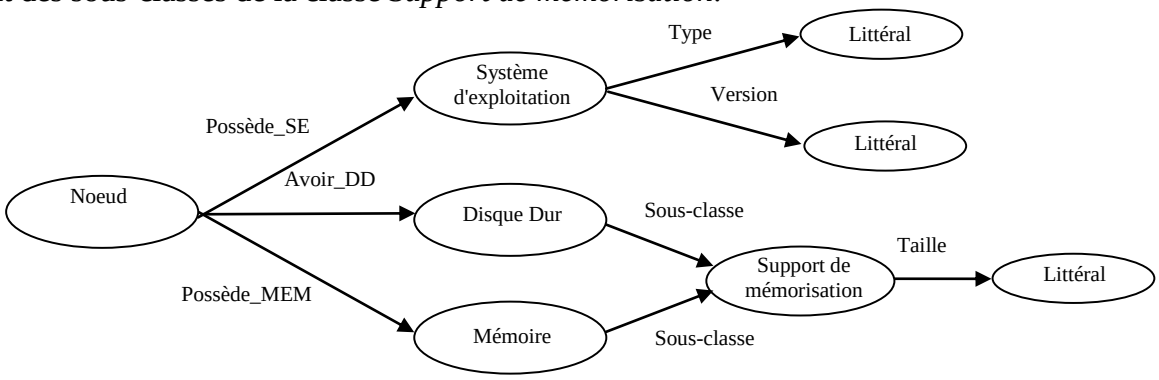


Figure 2.4. Schéma RDF

Composite Capability/Preference Profiles (CC/PP)

Le deuxième langage que nous étudions, est CC/PP. Proposé par le W3C, ce langage se base sur RDF pour fournir une description structurée des capacités du terminal et des préférences utilisateurs conformément au schéma suivant :

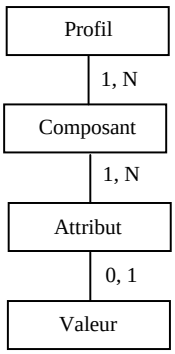


Figure 2.5. Schéma CC/PP

- Un profil CC/PP contient un ou plusieurs composants ;
- Un composant contient un plusieurs attributs. Chaque composant désigne une ressource comme le système d'exploitation ;
- Un attribut est une propriété du composant comme le type ou la version du système d'exploitation.

Ontology Web Language (OWL)

OWL est un langage de description à base d'ontologie permettant de décrire les classes de ressources et les relations entre elles. Il a été introduit en parallèle avec les premiers

travaux du web sémantique afin de faciliter le traitement et le raisonnement automatique sur les informations disponibles sur le web.

En pratique, le langage OWL est conçu comme une extension du langage RDF et de son schéma RDFS. OWL permet, grâce à sa sémantique formelle basée sur une fondation logique largement étudiée, de définir des associations plus complexes des ressources ainsi que les propriétés de leurs classes respectives. OWL définit trois sous-langages, du moins expressif au plus expressif : OWL-Lite, OWL-DL et OWL-Full [34].

Component Based Open Source Architecture for Distributed Telecom Applications (COACH)

COACH [35], est un projet de l'Information Society Technologies (IST) permettant de décrire l'environnement de déploiement. Il offre deux DTDs : la première se focalise sur le réseau physique. L'autre DTD se focalise sur les nœuds de déploiement. Les instances de ces DTDs forment les descripteurs utilisés par l'outil de déploiement développé dans le projet.

Le descripteur du réseau physique est défini essentiellement par :

- Des éléments de type *Nodes* permettant de déclarer les nœuds en incluant des références à leurs descripteurs de propriétés ;
- Des éléments de type *NodeLinks* permettant de définir les liens de communication entre les nœuds. Un lien est caractérisé par son nom, son type (ADSL, Ethernet, …), les deux nœuds dans les extrémités ainsi que d'autres propriétés comme la bande passante (Ko) et son pourcentage d'utilisation maximale ;
- Des services disponibles comme le service SSH. Chaque service est caractérisé par son nom, ainsi que d'autres attributs ;
- Des assemblages des composants déployés sur le nœud de déploiement. Les propriétés caractérisant un assemblage sont : l'identificateur, la description et l'état (installé, exécuté, inconnu).

Le descripteur du nœud du déploiement renseigne sur :

- La configuration logicielle comme le système d'exploitation et les programmes installés ;
- La configuration matérielle comme le processeur, le support de stockage, la mémoire.

2.5 Conclusion

Ce chapitre a présenté l'analyse et la classification des principales approches de déploiement et de reconfiguration. Nous avons d'abord présenté des définitions des principaux éléments clés appartenant au thème de cette thèse. Ensuite, nous avons expliqué les traits marquants de chaque approche de déploiement, montrant ainsi la complexité à la quelle il faut faire face, pour mettre en œuvre un système de déploiement dans le contexte du travail

collaboratif. Les modèles et les langages permettant d'automatiser le processus de déploiement et de reconfiguration, ont été exposés à la fin de ce chapitre.

En termes de bilan, le tableau 2.2 classifie selon les critères de déploiement identifiés, les principaux systèmes que nous avons présentés dans ce chapitre. Seul le critère performance n'a pas été utilisé car il est difficile d'obtenir des informations cohérentes sur ce point. Les stratégies proposées s'équilibrent entre les modes Push, Push-Pull et Pull. Les catégories de déploiement se partagent de façon équitable entre une approche statique et dynamique. Concernant les types de déploiement, une majorité des travaux portent sur le déploiement initial. Seuls les travaux des projets AMPROS et Planit proposent un déploiement ultérieur (ou reconfiguration). Une grande majorité de projets optent pour un contrôle centralisé. Seul CODEWAN adopte une approche décentralisée. Aucun des projets n'utilise de sensibilité active au contexte. En effet, la sensibilité active au contexte, plus difficile à mettre en œuvre, fait partie des thèmes de recherche récents et émergents liés à l'auto-adaptation.

	JWS	Software Dock	ORYA	CODEWAN	AMPROS	Planit
Stratégie	Pull	Push-Pull	Push	Push-Pull	Push	---
Catégorie	Statique	Statique	Statique	Dynamique	Dynamique	Dynamique
Types	Initial	Initial	Initial	Initial	Initial-Ultérieur	Initial-Ultérieur
Contrôle	Centralisé	Centralisé	Centralisé	Décentralisé	Centralisé	Centralisé
Sensibilité au contexte	---	Passif	Passif	----	Passif	Passif

Tableau 2.2. Principaux systèmes de déploiement

Les projets présentés dans l'état de l'art traitent du déploiement sans avoir de lien avec le domaine du travail collaboratif. A notre connaissance, il n'y a pas beaucoup de projets traitant le déploiement des applications dans les sessions collaboratives, à l'exception de Groove [36]. Ce dernier permet de constituer des communautés virtuelles. Il offre des fonctionnalités pour le déploiement des outils collaboratifs. Ses caractéristiques sont proches de celles de JWS. Bien que son architecture se base sur des interactions en mode Pair-à-Pair, un membre de la communauté choisit explicitement les applications à déployer sans garantir la compatibilité avec le contexte local ni l'interopérabilité avec les autres outils déployés sur les nœuds des participants. Le chapitre suivant décrit notre contribution. Il présente les éléments descriptifs proposés et détaille notre algorithme de déploiement et de reconfiguration.

Chapitre 3 Spécification de l'algorithme de déploiement

3.1 Introduction

Ce chapitre décrit la solution de déploiement et de reconfiguration que nous avons développée dans le cadre de cette thèse. Nous commençons par illustrer les phases communes identifiées dans le chapitre précédent. Ensuite, nous décrivons notre première contribution pratique qui consiste en trois modèles permettant d'automatiser le processus de déploiement. Ces modèles ont été essentiellement choisis pour leur simplicité et leur facilité d'utilisation. La troisième partie est dédiée à l'algorithme de déploiement et de reconfiguration. Enfin, nous détaillons les différentes étapes de l'algorithme à travers un exemple illustratif.

3.2 Phases du déploiement sensible au contexte

L'étude des travaux liés au déploiement nous a permis d'identifier quatre phases indispensables pour le déploiement (figure 3.1) :

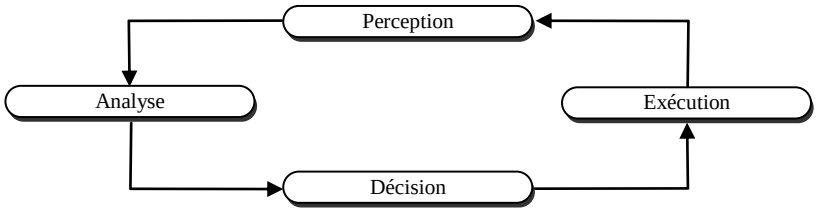


Figure 3.1. Phase du déploiement sensible au contexte

Perception

La phase de perception sert à récupérer les données nécessaires au déploiement. Elle permet de percevoir le contexte auquel est sensible l'application à déployer. Cette phase peut être activée par des sondes logicielles à différents moments, ou bien de façon périodique.

Dans notre cas il s'agit non seulement du contexte local formé par l'environnement d'exécution mais aussi du contexte global c'est-à-dire des applications déployées sur les nœuds voisins. Les données de contexte doivent être fournies sous une forme qui puisse être interprétée par une couche logicielle pour pouvoir être traitées de façon automatique. La phase de collecte d'informations permet finalement de découvrir la liste des applications candidates pour le déploiement.

Analyse

La deuxième phase permet d'analyser les informations recueillies pendant la phase de collecte d'informations. La comparaison entre les ressources exigées par les applications et les ressources offertes par les nœuds de déploiement se fait pendant cette phase.

Décision

Une étape cruciale de l'algorithme de déploiement concerne la phase de prise de décision. En se basant sur les informations récoltées et analysées, l'algorithme de déploiement décide des applications à rapatrier.

Exécution

L'objectif de cette phase est d'appliquer la décision prise pendant la phase précédente. Elle concerne donc le rapatriement des applications choisies vers le nœud de déploiement. Elle peut englober d'autres opérations comme la suppression de certaines applications.

Ces phases sont successives et forment le cycle de vie du déploiement. Une fois que la phase d'exécution a été faite, le cycle se répète pour suivre l'évolution du contexte.

Afin de supporter ces phases, il faut avoir :

- Des structures de données pour décrire les besoins des utilisateurs, les caractéristiques des applications et des nœuds qui les supporteront ;
- Un ensemble de procédures telles que la vérification de la compatibilité entre l'application et son environnement d'exécution ;
- Une plate-forme pour supporter l'algorithme de déploiement et faciliter les échanges de données (descripteurs et applications) entre les nœuds.

La section suivante décrit notre approche pour les structures de données. La section 3.3 regroupe les procédures nécessaires. La conception de la plate-forme, plus complexe, fait l'objet du chapitre 4.

3.3 Modèles pour automatiser le déploiement

Pour pouvoir fournir un support automatisé et générique, il est nécessaire de se baser sur une approche orientée modèle. Nous avons défini trois modèles :

- Un modèle d'application qui donne les exigences des applications, c'est-à-dire les ressources requises pour le bon fonctionnement de l'application.
- Un modèle de session qui décrit de façon abstraite la structure de la session ;
- Un modèle du nœud du déploiement qui décrit le contexte local, c'est-à-dire les ressources offertes par le nœud de déploiement ;

Ces modèles génériques permettent d'abstraire les informations de déploiement. Le terme générique se justifie par le fait que ces modèles sont indépendants des applications à déployer, du domaine de la session, et du nœud de déploiement.

Les caractéristiques de chacun de ces modèles ont été décrites en utilisant le langage UML [37], universellement utilisé pour spécifier des systèmes. L'implantation de chaque modèle se fait en utilisant le langage XML [38]. Dans la suite, nous allons détailler chaque modèle.

3.3.1 Modèle d'application

Le modèle d'application proposé permet à un service de déploiement de choisir une application plutôt qu'une autre. Trois contraintes influencent ce choix :

- Le rôle d'un participant. Les applications doivent couvrir le rôle du participant. Par exemple, en utilisant un outil de partage de document, l'enseignant diffuse son cours vers les étudiants. En même temps, cet enseignant reçoit les commentaires sous forme textuelle en utilisant un autre outil de dialogue.
- Compatibilité avec l'environnement d'exécution. Le nœud du déploiement doit offrir les ressources nécessaires pour que les applications puissent fonctionner correctement.
- Interopérabilité avec les applications déjà déployées sur les nœuds des voisins. Les membres d'une session collaborative s'échangent des informations. Ainsi, les applications doivent être interopérables pour qu'elles puissent communiquer sans ambiguité et opérer ensemble.

Afin de respecter toutes ces contraintes. Nous notons qu'une application peut avoir plusieurs implantations. Une implantation d'application est toujours complète, dans le sens où elle couvre tous les composants de l'application. A la différence du déploiement d'applications à base de composants qui se charge de trouver les implantations de chaque composant et les nœuds adéquats, nous déployons uniquement l'implantation qui représente l'application en totalité sur le nœud se connectant à la session. Nous avons mis l'accent sur une classification des applications par rapport au traitement des flux de données. Nous retrouvons donc les catégories suivantes :

Application monolithique monomédia unidirectionnelle

Cette catégorie implique que l'application est une seule entitée monolithique conçue pour produire ou pour consommer un seul type de données. La figure 3.2 illustre les différentes possibilités d'instanciation d'une application appartenant à cette catégorie.

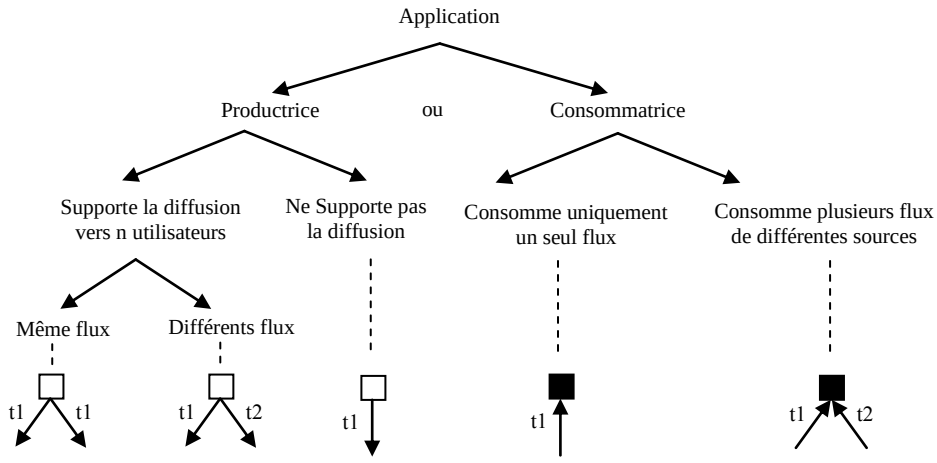


Figure 3.2. Possibilités d'instanciation (1ère catégorie)

Une application productrice peut supporter ou non la diffusion vers plusieurs participants. Dans le cas où elle peut supporter la diffusion, on distingue deux situations : (1)

l'application envoie le même flux vers n participants. C'est le cas d'un enseignant qui diffuse son cours vers les étudiants. (2) l'application envoie différents flux vers n participants. C'est le cas d'un étudiant qui émet des flux textuels vers les autres étudiants. Si l'application ne supporte pas la diffusion, alors elle produit uniquement un seul flux de données vers un seul participant.

Dans la deuxième sous-catégorie, on distingue deux situations : (1) l'application est capable de consommer un seul flux de données. Par exemple, un étudiant reçoit le cours diffusé par l'enseignant. (2) l'application est capable de consommer plusieurs flux de données provenant de différentes sources. Par exemple, un étudiant reçoit les messages textuels envoyés par les autres étudiants.

Application composite monomédia

Cette catégorie comprend des applications formées par une entité logicielle composites qui se déclinent en deux parties, une partie productrice et une partie consommatrice. Une telle application n'est capable de traiter qu'un seul type de données. L'application peut être instanciée en rôle producteur, consommateur ou les deux à la fois (figure 3.3).

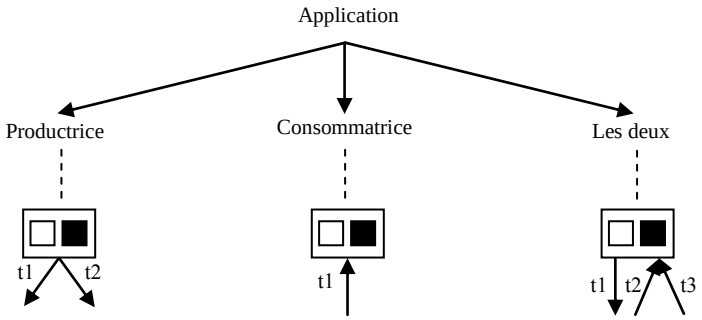


Figure 3.3. Possibilités d'instanciation d'une application (2ère catégorie)

Par exemple, une application de dialogue qui produit et qui reçoit des flux textuels fait partie de cette catégorie.

Application composite multimédia

C'est le cas le plus général. Une application de cette classe est composée d'une collection d'entités logicielles qui manipulent différents types de données et qui se déclinent chacune en une partie productrice et une partie consommatrice. Une application de dialogue sophistiquée qui permet d'échanger du texte et de la parole, fait partie de cette catégorie.

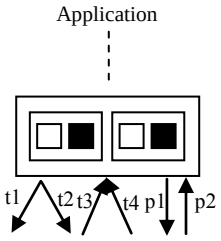


Figure 3.4. Possibilité d'instanciation (3ème catégorie)

Une représentation formelle de ce modèle est donnée par la figure suivante :

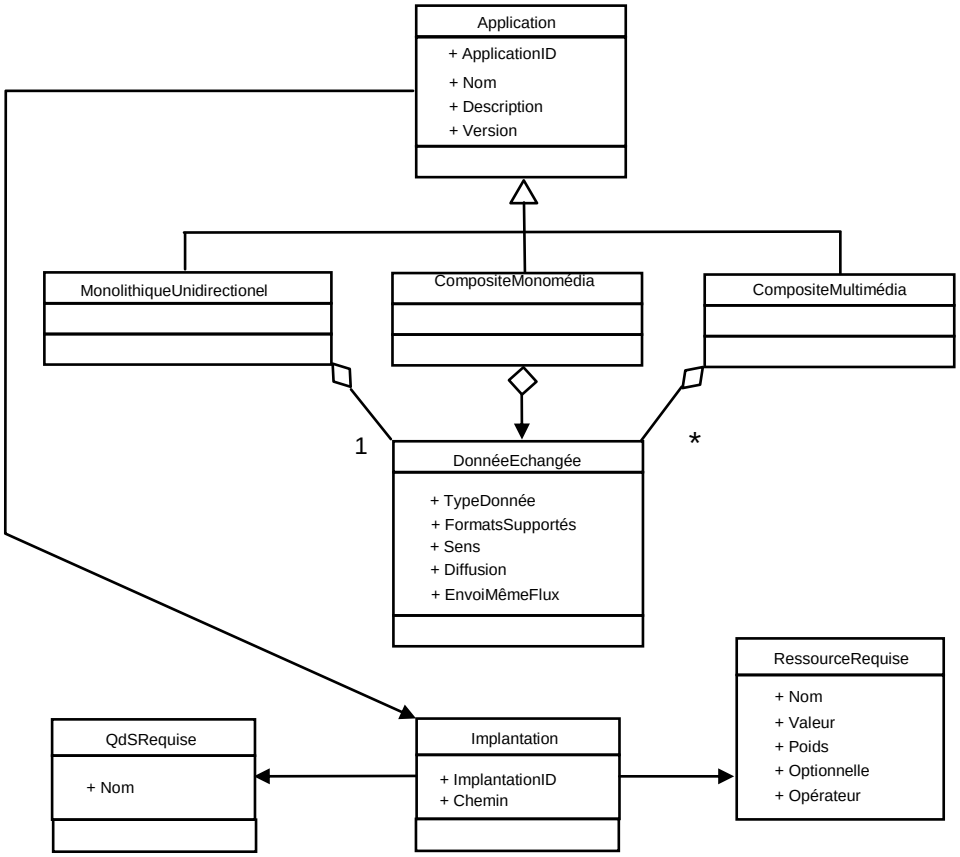


Figure 3.5. Représentation UML du modèle d'application

D'après le modèle, une application peut avoir plusieurs implantations. Une implantation d'application est toujours complète, dans le sens où elle couvre tous les composants de l'application. A la différence du déploiement d'applications à base de composants qui se charge de trouver les implantations de chaque composant et les nœuds adéquats, nous déployons uniquement l'implantation qui représente l'application en totalité sur le nœud se connectant à la session.

Pour qu'une implantation d'application fonctionne correctement, il est nécessaire que le nœud qui va l'accueillir possède les ressources et les services requis pour son bon fonctionnement. Un descripteur *ImplantationID* est associé à chaque implantation d'application. Comme décrit dans l'état de l'art, nous trouvons dans la littérature des langages comme OSD, JNLP et DSD, permettant de décrire les applications et leurs implantations. Ils contiennent essentiellement les caractéristiques de l'application. Dans notre cas, les attributs et les propriétés encapsulés dans le descripteur *ImplantationID* se divisent en deux familles :

- **Qualité de service requise**. Si une implantation d'application utilise un service non fonctionnel tel que la gestion des transactions, et qu'elle possède le code spécifique réalisant ce service, alors elle peut être déployée sur n'importe quel nœud. Par contre si l'implantation d'application ne possède pas le code réalisant ce service mais qu'elle a absolument besoin de ce service, alors elle ne doit être déployée que sur un nœud offrant le service en question.

- **Ressources requises**. Une implantation d'application nécessite des ressources pour s'exécuter comme la puissance de traitement, la taille minimale sur le disque dur, le type du système d'exploitation. Certaines ressources présentent des contraintes fortes, d'autres non (attribut optionnel).

Lors de la sélection d'une implantation d'application, le nombre d'instanciations varie de 1 à N selon le nombre de flux manipulés par l'application associée, et requis par la session. Ce nombre d'instances sera ainsi déterminé par la classe d'application que l'on déploie et par la capacité de l'application à traiter plusieurs flux de données (champ *Diffusion*).

La figure 3.6 montre la représentation XML d'une application monolithique monomédia unidirectionnelle de dialogue textuel. Cette application produit et consomme un flux textuel supportant les formats ascii et unicode. Une seule implantation compose l'application. Le service de persistance est nécessaire pour le fonctionnement de cette implantation. Les ressources requises sont : une fréquence de microprocesseur supérieur à 1.00 GHz et une mémoire vive d'au moins 64 Mo. Ces deux ressources sont obligatoires. Par contre la connexion Internet de 56 ko est optionnelle.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<CompositeMonomedia>
	<ApplicationID>md5-d2ab42830caa7a02b36d96a46dd44b8d</ApplicationID>
	<Nom>Dialogue</Nom>
	<Description>Application de dialogue textuel</Description>
	<Version>1.0</Version>
	<DonnéesEchangés>
		<DonnéeEchangé>
			<TypeDonnée>Texte</TypeDonnée>
			<FormatsSupportés>
				<Format>ascii</Format>
				<Format>unicode</Format>
			</FormatsSupportés>
			<Sens>producteur</Sens>
			<Diffusion>vrai</Diffusion>
			<EnvoiMêmeFlux>vrai</EnvoiMêmeFlux>
		</DonnéeEchangé>
		<DonnéeEchangé>
			<TypeDonnée>Texte</TypeDonnée>
			<FormatsSupportés>
				<Format>ascii</Format>
				<Format>unicode</Format>
			</FormatsSupportés>
			<Sens>consommateur</Sens>
			<Diffusion>faux</Diffusion>
			<EnvoiMêmeFlux>faux</EnvoiMêmeFlux>
		</DonnéeEchangé>
	</DonnéesEchangés>
	<Implantations>
		<Implantation>
			<ImplantattionID>md5-d9e90367d6bd2edb8d5f2551c2feb202</ImplantationID>
			<Nom>impl1</Nom>
			<Chemin>c:\repository\TalkP+C.jar</Chemin>
			<QdSRequise>persistency</QdSRequise>
			<RessourcesRequises>
				<Ressource>
					<Nom>CPU</Nom>
					<Type>Integer</Type>
					<Valeur>1.00</Valeur>
					<Opérateur>inferiororequal</Opérateur>
```

```
                                        <Optionnelle>faux</Optionnelle>
                                        <Poids>2</Poids>
                                </Ressource>
                                <Ressource>
                                        <Nom>RAM</Nom>
                                        <Type>Integer</Type>
                                        <Valeur>64</Valeur>
                                        <Opérateur>inferiororequal</Opérateur>
                                        <Optionnelle>faux</Optionnelle>
                                        <Poids>2</Poids>
                                </Ressource>
                                <Ressource>
                                        <Nom>Internet</Nom>
                                        <Type>Integer</Type>
                                        <Valeur>56</Valeur>
                                        <Opérateur>inferiororequal</Opérateur>
                                        <Optionnelle>vrai</Optionnelle>
                                        <Poids>3</Poids>
                                </Ressource>
                        </RessourcesRequises>
                </Implantation>
        <Implantations>
<CompositeMonomedia>
```

Figure 3.6. Représentation XML d'une application de dialogue textuel

3.3.2 Modèle de session

Un modèle basé sur un graphe étiqueté orienté a été d'abord introduit par T. Villemur [39], puis repris par L. Rodriguez [40] [41] pour représenter les interactions entre les utilisateurs d'une session collaborative. Nous avons étendu ce modèle pour l'adapter au contexte du déploiement collaboratif [42], [43]. Ainsi, les nœuds du graphe représentent les terminaux de déploiement avec leurs utilisateurs connectés. Les relations entre nœuds (traduites par les flèches), expriment les échanges de données entre terminaux. Les étiquettes sur les flèches sont composées de deux champs. Le premier champ donne le type du flux de donnée échangé (vidéo, audio, texte, …). Le deuxième champ donne le nom du flux de donnée, permettant ainsi de séparer des flux différents mais qui échangent un même type de média. La figure 3.7 représente un exemple d'une session qui comprend quatre utilisateurs représentés par les nœuds n1, n2, n3 et n4. Ces utilisateurs manipulent des flux audio et vidéo entre eux.

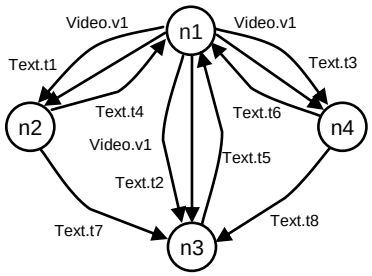


Figure 3.7. Instance du modèle de session

Une représentation formelle de ce modèle est donnée par la figure 3.8.

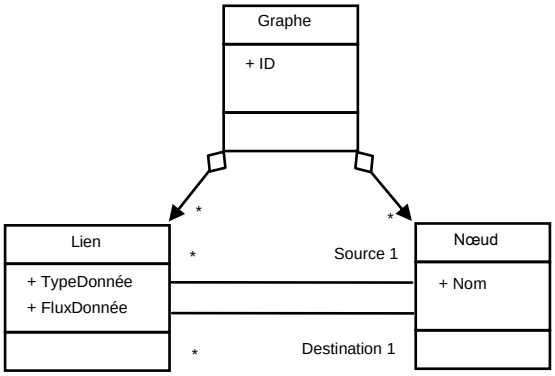


Figure 3.8. Représentation UML du modèle de session

Ce modèle s'appuie sur des relations de type producteur/consommateur de données. Nous avons volontairement orienté le modèle dans ce sens car nous désirons traiter et représenter des échanges de données dans des sessions de travail synchrones et interactives. De telles sessions manipulent des flux de données interactifs (par exemple vidéo, audio temps réel) qui se représentent naturellement par des relations entre utilisateurs.

La figure 3.9 donne le code XML qui correspond à l'instance de la session de la figure 3.7.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Graphe id="g-0001">
        <Nœuds>
                <Nœud>n1</Nœud>
                <Nœud>n2</Nœud>
                ...
        </Nœuds>
        <Liens>
                <Lien Source="n1" Destination="n2">
                        <TypeDonnée>Text</TypeDonnée>
                        <FluxDonnée>t1</FluxDonnée>
                </Lien>
                ...
        </Liens>
</Graph>
```

Figure 3.9. Représentation XML d'une session de quatre utilisateurs

3.3.3 Modèle de nœud de déploiement

Un nœud de déploiement constitue l'environnement d'exécution des applications. Un nœud est défini par un identificateur *ID*. L'attribut *Nom* indique la désignation du nœud. L'attribut *Type* donne la catégorie (PC, PDA, ...). Le modèle de nœud de déploiement doit contenir les informations sur (i) les *Ressources offertes*, qui contiennent les caractéristiques logicielles du nœud comme le système d'exploitation et les caractéristiques matérielles comme la mémoire, (ii) la *QdS offerte (Qualité de Service)* qui contient la liste des services logiciels de ce nœud. Notre proposition est proche des descriptions de contextes d'exécution faites dans les travaux décrits dans l'état de l'art (RDF, CC/PP, OWL et COACH). Le modèle que nous proposons est décrit par le diagramme de classe UML de la figure 3.10. Chaque nœud de déploiement possède une instance de ce modèle décrivant son état.

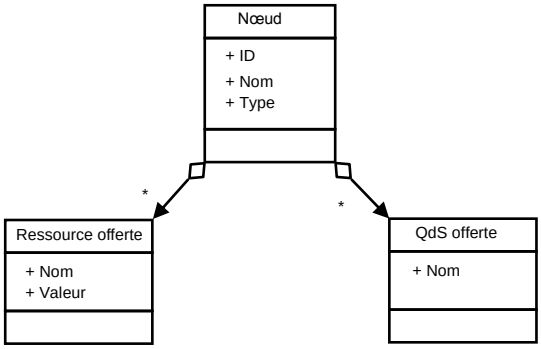


Figure 3.10. Représentation UML du modèle de nœud de déploiement

La figure 3.11 montre la représentation XML d'un nœud de déploiement qui supporte les services de persistance et de sécurité. Ce nœud offre également les ressources suivantes : mémoire virtuelle de 512 Mo et fréquence processeur de 2.80 GHz.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Nœud>
        <ID>md5-044c0f755d7e410a0da4c7d2b1a52263</id>
        <Nom>n1</name>
        <type>PC</type>
        <relevantcontext>
                <offeredqos>persistency,security</offeredqos>
                <Offeredresources>
                        <CPU>2.80</CPU>
                        <RAM>512</RAM>
                </Offeredresources>
        <relevantcontext>
</Nœud>
```

Figure 3.11. Représentation XML du nœud de déploiement n1

En comparant les informations sur le nœud de déploiement avec les ressources requises pour une implantation, il est possible de vérifier la compatibilité avec le nœud, ce qui permet de sélectionner une implantation plutôt qu'une autre et donc d'autoriser ou de refuser son rapatriement.

3.4 Algorithme de déploiement et de reconfiguration

3.4.1 Vue générale

Chaque nœud exécute l'algorithme de déploiement et de reconfiguration. Ce dernier reçoit l'instance du modèle de session décrivant l'état que l'on veut atteindre. En utilisant cette instance, l'algorithme tente de fournir au nœud sur lequel il s'exécute, les applications nécessaires conformément au modèle de session, tout en respectant les contraintes de compatibilité et les contraintes d'interopérabilité avec les applications déployées sur les nœuds voisins.

L'algorithme permet le déploiement initial et le déploiement ultérieur (reconfiguration). L'entrée d'un utilisateur dans la session entraîne l'exécution du déploiement initial sur son nœud. Par la suite, si des changements surviennent au niveau des rôles des participants à la

session, alors le déploiement ultérieur s'exécute sur les nœuds impliqués dans l'évolution de la session (figure 3.12).

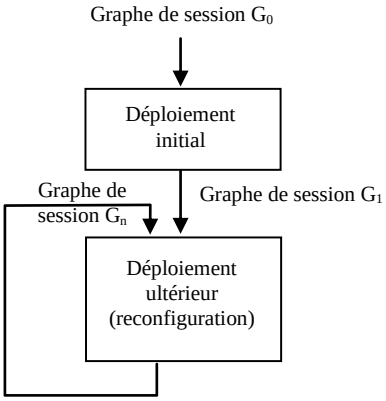


Figure 3.12. Vue générale de l'algorithme de déploiement et de reconfiguration

On trouve une corrélation entre le monde réel formé par les nœuds de déploiement et le monde abstrait formé par le modèle de session. Lors de l'évolution dans le temps, provoquée par les évènements extérieurs comme les entrées / sorties d'utilisateurs ou les changements de rôles, il est nécessaire de maintenir une évolution conjointe cohérente entre le monde réel et le modèle. L'algorithme de déploiement et de reconfiguration fait partie de la chaîne de services qui contribuent à cette évolution conjointe.

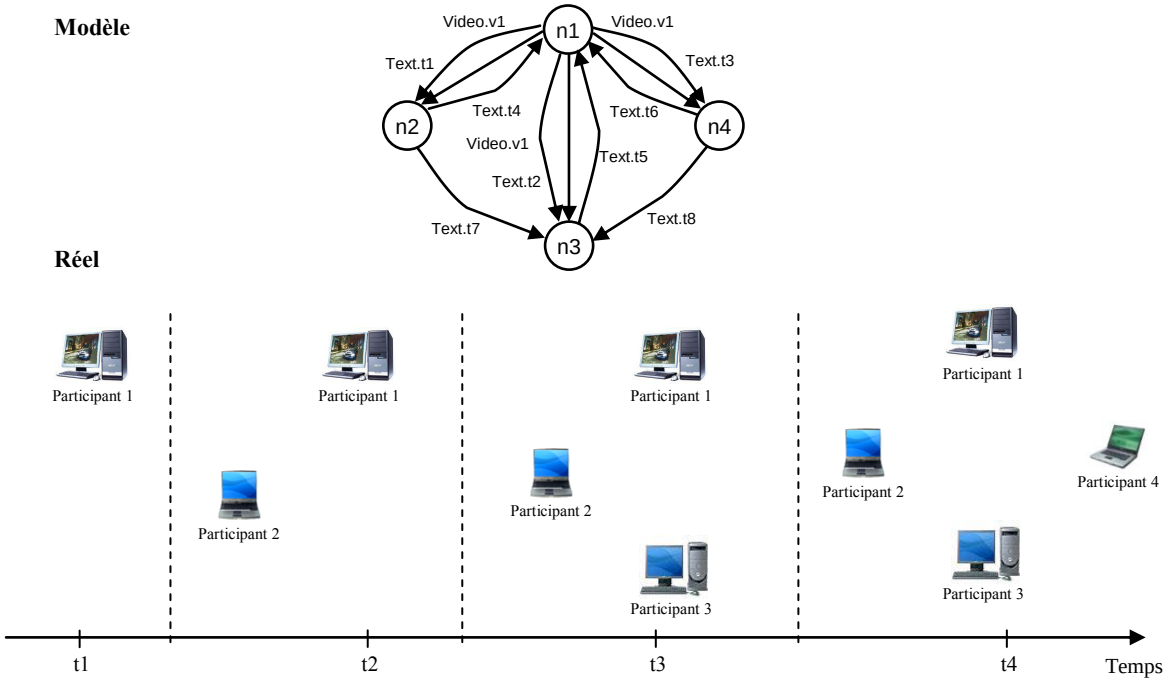


Figure 3.13. Déploiement initial

La figure 3.13 illustre le démarrage de la session. Les entrées successives des participants 1 à 4 provoquent le déploiement initial. Par la suite, lorsque le modèle de session évolue, les participants impliqués dans cette évolution, vont pouvoir exécuter les opérations du déploiement ultérieur (figure 3.14).

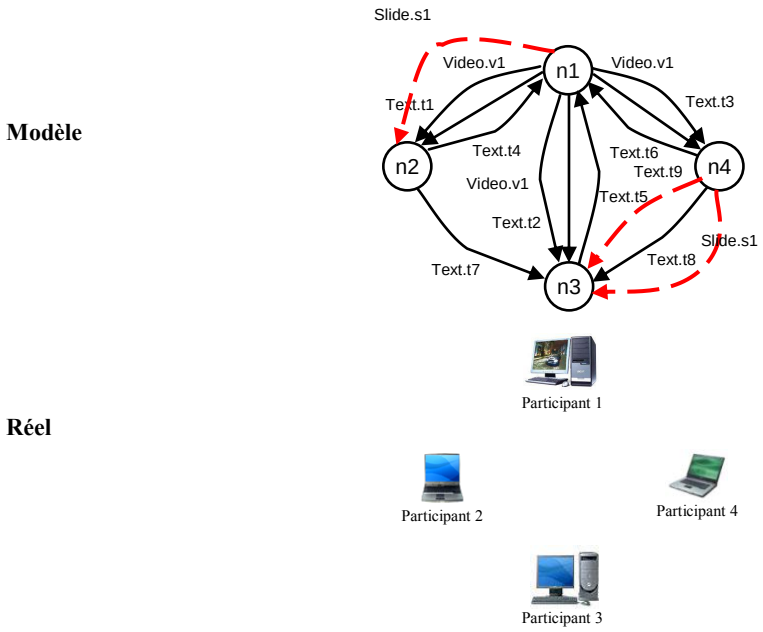


Figure 3.14. Déploiement ultérieur (reconfiguration)

Plusieurs facteurs rendent la reconfiguration difficile à résoudre. D'une part, les applications déployées sont hétérogènes. D'autre part, la session réunit plusieurs participants, ayant chacun plusieurs applications. Accomplir un changement global de la session dans ce contexte, est une tâche qui n'est pas facile.

Même sans prendre en compte les deux facteurs précédents, l'ajout et la suppression des applications restent délicats à cause des interdépendances entre les applications. Il est important de garder la cohérence de la session en effectuant ces changements. L'état de la session avant et après la reconfiguration doit être cohérent pour que les membres puissent travailler sans problème. De plus, les contraintes sur les ressources, les dépendances entre les applications, et les restrictions temporelles rendent le problème de la reconfiguration encore plus difficile [2]. En effet, il ne faut pas que la reconfiguration prenne beaucoup de temps car, durant son exécution, l'activité collaborative ne peut pas se dérouler normalement, voire même est suspendue.

3.4.2 Déploiement initial

Introduction

Le déploiement initial permet au nœud qui se connecte à la session d'avoir les applications qui lui font défaut. L'algorithme accepte en entrée l'instance du modèle de session sous forme d'un fichier XML (figure 3.15). En le parcourant, l'algorithme détermine les besoins de l'utilisateur et exécute les instructions nécessaires pour répondre à ces besoins. Le diagramme suivant illustre le déroulement de l'algorithme dans le cas du déploiement initial.

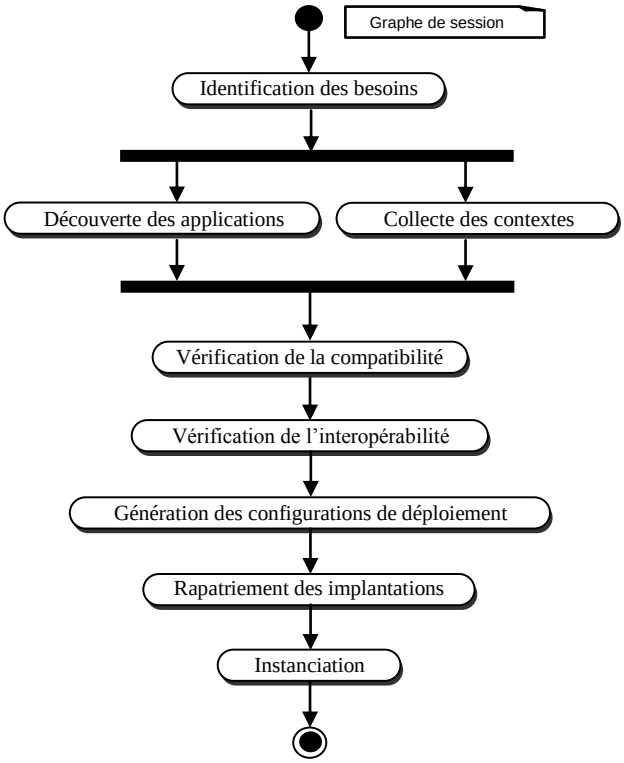


Figure 3.15. Etapes du déploiement initial

Certaines procédures se déroulent dans des processus parallèles séparés permettant ainsi de réduire le temps du déploiement.

Etapes du déploiement initial

Identification des besoins

L'algorithme de déploiement reçoit en entrée le graphe décrivant la session. En le parcourant, il détermine :

- L'ensemble des couples {type de données, sens}. Chaque couple donne les caractéristiques nécessaires pour un flux de données requis. Le **type** représente le type de données échangées entre deux utilisateurs, le **sens** indique si le flux est envoyé (production) ou reçu (consommation). Cette information est utilisée pour la découverte des applications.

- Les flux de données ainsi que la liste des nœuds voisins de chaque flux. La relation {Flux de données → Voisins} maintient le lien entre un flux de données et les voisins du nœud qui utilisent ce flux. La liste des nœuds voisins est utilisée pour récupérer le contexte global. En effet, chaque nœud dispose d'un descripteur décrivant les implantations déployées. L'ensemble de ces descripteurs constitue le *contexte global* qui sert à vérifier l'interopérabilité entre applications. Ceci est utile pour déterminer le nombre d'instanciation des applications rapatriées.

Besoin	{Type de données, sens}	Flux de données → Voisins
b_1	{Vidéo, Production}	v1→ {n2, n3, n4}
b_2	{Texte, Production}	t1 → {n2}
		t2 → {n3}
		t3 → {n4}
b_3	{Texte, Consommation}	t4 → {n2}
		t5 → {n3}
		t6 → {n4}

Tableau 3.1. Besoins du nœud n1

Le tableau 3.1 décrit les besoins obtenus pour le nœud n1 de la figure 3.7. Ainsi, b_1 (colonne "Besoins") identifie le type Vidéo en production. Les voisins de n1 par rapport à ce besoin sont {n2, n3, n4} (déduits de la colonne "Flux de données → Voisins").

Découverte des applications

La découverte des applications est une fonction très importante dans le processus de déploiement. Etant donné un type de donnée et un sens de communication, elle permet de localiser les descripteurs des applications qui répondent à ces critères de recherche. La découverte est effectuée de façon dynamique c'est-à-dire pendant le déploiement.

Pour le nœud n1, les critères de recherche seront b_1, b_2 et b_3. A la fin de cette étape, le nœud se retrouve avec une liste des descripteurs d_i des applications candidates. A titre d'exemple, nous supposons que trois applications $\{a_1, a_2, a_3\}$ couvrent le besoin b_1. Lors de la découverte, on rapatrie les descripteurs $\{d_1, d_2, d_3\}$ qui leur sont associés. Le détail de couverture est donné par le tableau 3.2.

Besoin	Descripteurs d'applications découverts
b_1	$\{d_1, d_2, d_3\}$
b_2	$\{d_4, d_5\}$
b_3	$\{d_4\}$

Tableau 3.2. Descripteurs d'applications découverts pour le nœud n1

Nous remarquons que pour les besoins b_2 et b_3, nous avons trouvé une application a_4, de descripteur d_4, appartenant à la catégorie des applications composites monomédia, qui prend en charge l'envoi et la réception de flux de données de type texte.

Collecte des contextes

Pour pouvoir choisir les implantations appropriées, l'étape de collecte des contextes permet :

- d'une part de récupérer le descripteur de contexte (local) du nœud sur lequel les implantations vont être installées,
- et d'autre part, de récupérer le descripteur décrivant les applications (plus précisément les implantations) déjà déployées sur un nœud voisin pour la session en cours. Ceci est utile pour vérifier l'interopérabilité avec les applications distantes.

En repartant de la structure courante de la session donnée par la figure 3.2, le nœud n1 utilise non seulement son propre descripteur de contexte mais aussi il récupère les descripteurs décrivant les applications déployées sur les nœuds n2, n3 et n4 avec lesquels il est en contact.

Vérification de la compatibilité

A partir de la liste retournée suite à l'opération de découverte, l'algorithme de déploiement choisit les implantations compatibles au contexte local du nœud sur lequel elles vont être installées.

Etant donné que chaque application a_i se compose d'un ensemble d'implantations I_1 .. I_n, l'algorithme procède de la manière suivante : pour chaque descripteur découvert, l'algorithme parcourt les parties relatives aux implantations possibles de l'application et compare les ressources et les paramètres de QdS requis avec ceux contenus dans le descripteur du nœud. S'ils sont incompatibles entre eux, alors l'implantation est rejetée. Cette partie de l'algorithme se présente de cette façon :

```
1.  Pour chaque besoin b_i identifié dans la première phase faire
2.      Pour chaque descripteur d'application d_j vérifiant b_i faire
3.          Pour chaque implantation I_k contenue dans d_j faire
4.              Si non compatible (I_k , descripteur de terminal)
5.                  Alors élimer I_k
6.          Fin
7.      Fin
8.  Fin
```

Pour vérifier si une implantation respecte ou non les caractéristiques d'un nœud, l'algorithme utilise la fonction *compatible* définie comme suit :

```
1.  compatible ← vrai
2.  Tant qu'il reste des ressources requises et (compatible == vrai)
3.      Réinitialiser la liste des ressources offertes
4.      trouvée ←faux
5.      Tant qu'il reste des ressources offertes et (non trouvée)
6.          Si nom de la ressource requise = nom de la ressource offerte
7.              Alors
8.                  trouvée ← vrai
9.          Si valeur de la ressource requise ne correspond pas à la valeur de la ressource offerte
10.             Alors
11.                 Si ressource requise est non optionnelle
12.                     Alors compatible ←faux
13.     Fin
14.     Si non trouvée
15.         Alors Si ressource requise est non optionnelle
16.             Alors compatible ← faux
17. Fin
18. Retourne compatible
```

A l'issue de cette étape, seules les implantations compatibles avec le contexte local sont gardées, les autres sont éliminées.

Le tableau 3.3 donne un résultat possible de cette étape pour le besoin b_2 {Texte, Production} du nœud n1.

Besoin	**Avant filtrage**	**Après filtrage**
b_2	d_4→ {I_0, I_1} d_5 → {I_2, I_3, I_4, I_5}	d_4→ {I_1} d_5 → {I_2, I_3, I_4}

Tableau 3.3. Résultat possible pour le besoin b_2 du nœud n1

Pour le besoin b_2 (colonne "Besoin"), avant la vérification de la compatibilité locale, six implantations forment la liste des implantations candidates (colonne "Avant filtrage"). A l'issue de cette phase, il ne reste plus que quatre implantations candidates (colonne "Après filtrage").

Vérification de l'interopérabilité

L'interopérabilité est la capacité pour deux ou plusieurs applications, se trouvant éventuellement sur différentes machines, de communiquer, d'échanger des données et d'utiliser ces données échangées [44].

Du fait que certaines applications peuvent ne pas être interopérables avec les autres applications sur les nœuds en relation directe avec eux, un deuxième filtrage est donc nécessaire pour assurer l'interopérabilité distribuée.

En utilisant les informations récupérées lors de la phase de collecte de contextes, l'algorithme détermine pour chaque implantation, les nœuds voisins qui offrent les applications interopérables avec elle :

1. *Pour chaque besoin b_i identifié lors de la première phase faire*
2. *Pour chaque implantation I_j vérifiant b_i faire*
3. *Pour chaque nœud voisin n_k restreint à b_i faire*
4. *si interopérable(I_j, n_k, b_i^{-1})*
5. *alors ajouter n_k à la liste des nœuds interopérables de I_j*
6. *Fin*
7. *Fin*
8. *Fin*

b_i^{-1} exprime le besoin opposé. C'est-à-dire, si b_2 = {Texte, Production}, alors b_2^{-1} = {Texte, Consommation}.

Dans la figure 3.2, les nœuds voisins au nœud n1 restreints au besoin b_2 sont {n2, n3, n4}. Nous avons trouvé après la phase de découverte et de vérification des contraintes de compatibilité locale que les implantations candidates sont {I_1, I_2, I_3, I_4}.

Si I_1 produit un flux Texte qui supporte le format Unicode, alors la fonction *interopérable* retourne Vrai si cette implantation est interopérable avec celle déployée sur chaque nœud n2, n3 et n4 traitant le type de donnée Texte en consommation. Par exemple, si le nœud n2 contient l'implantation I traitant un flux de type Texte en consommation et supportant le format Unicode alors *interopérable(I_1, n2, b_2^{-1})* retourne Vrai. Dans le cas contraire, elle retourne Faux et les implantations ne sont pas interopérables.

Par exemple, à l'issue de cette étape, les implantations candidates pour satisfaire le besoin b_2 du nœud n1 sont celles décrites par le tableau 3.4 :

Implantation	Nœuds interopérables
I_1	{n2, n3, n4}
I_2	{n3}
I_3	{n3, n4}
I_4	{ }

Tableau 3.4. Implantations candidates pour le besoin b_2 du nœud n1

Génération des configurations de déploiement

Cette phase constitue la partie la plus complexe de notre algorithme. Elle représente donc notre contribution majeure. Le but est de générer les configurations de déploiement valides. La difficulté réside dans le fait que pour un besoin spécifique, par exemple Texte en production, une implantation peut être interopérable avec un nœud voisin mais pas avec un autre. Il faut donc chercher une autre implantation interopérable avec ce deuxième nœud. De plus, si le flux de données est le même pour les deux voisins (en cas de diffusion simultanée), alors il faut absolument que l'implantation à déployer soit interopérable avec toutes les implantations déployées chez les voisins, sinon elle ne doit pas être prise en compte. Les

implantations réunies doivent former une configuration de déploiement valide permettant de couvrir la solution sur le nœud.

Pour chaque couple {type, sens}, L'algorithme commence d'abord par éliminer :

1. les implantations qui ne sont interopérables avec aucun nœud voisin.
2. les implantations qui ne sont pas interopérables avec tous les nœuds voisins consommant le même flux de données.
3. les implantations qui doivent produire le même flux de données pour plusieurs nœuds voisins, alors qu'elles ne permettent pas la diffusion.

Avec les implantations restantes, l'algorithme génère toutes les combinaisons possibles formant les configurations de déploiement. Si n est le nombre des implantations candidates, alors les configurations générées sont au nombre de $2^n - 1$.

Pour chaque configuration, l'algorithme vérifie si elle satisfait tous les nœuds voisins. Dans ce cas, elle est marquée comme valide.

Pour bien expliquer cette procédure, nous considérons différents cas possibles de modèles de sessions. Ainsi, dans le modèle de gauche, trois flux Texte différents sont produits par le nœud n1. Dans celui du centre, n1 diffuse le flux t1 pour {n2, n3} et t2 vers {n4}. Dans le modèle de droite, un même flux t1 est diffusé par n1 vers {n2, n3, n4}. En partant des implantations {I_1, I_2, I_3} (I_4 est éliminée à l'issue de l'étape 1 car elle n'est interopérable avec personne), l'algorithme garde les implantations candidates pour n1 dans chaque structure de session. Le résultat est décrit dans le tableau 3.5 (étape 2) :

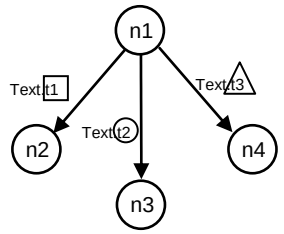
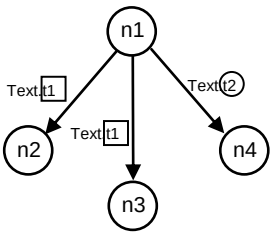
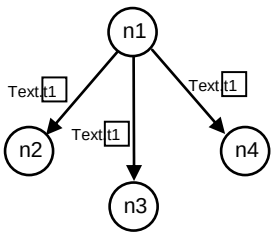
Modèles de session			
Implantations candidates pour le nœud n1	I_1, I_2 et I_3 sont sélectionnées puisque chacun possède dans sa liste des voisins interopérables (Tableau V), au moins un voisin figurant parmi ceux associés au nœud n1.	I_1 et I_3 sont sélectionnées. I_2 est éliminée puisqu'elle n'est pas interopérable avec n2 alors quelle doit produire le même flux t1 pour n2 et n3.	Seule I_1 est sélectionnée. Les autres sont éliminées puisqu'elles ne sont pas interopérables avec tous les voisins pour le même flux t1.

Tableau 3.5. Elimination des implantations inappropriées

Le modèle de la figure 3.7 restreint au besoin b_2 du nœud n1 correspond au premier cas dans le tableau 3.5. Seules les implantations I_1, I_2 et I_3 sont retenues. Avec ces trois implantations, l'algorithme génère sept configurations de déploiement possibles. Ensuite, l'algorithme examine les différents cas pour ne garder que les configurations valides (lignes n° 1, 4, 5 et 7) qui répondent à toutes les contraintes locales et globales :

N°	**Configuration**	**n2**	**n3**	**n4**
1	I_1	✕	✕	✕
2	I_2		✕	
3	I_3		✕	✕
4	I_1, I_2	✕	✕	✕
5	I_1, I_3	✕	✕	✕
6	I_2, I_3		✕	✕
7	I_1, I_2, I_3	✕	✕	✕

Tableau 3.6. Configurations de déploiement générées

Dans certains cas de figure, il est possible qu'après la vérification des contraintes de compatibilité locale et d'interopérabilité, l'algorithme ne trouve aucune configuration valide. Cela veut dire qu'avec les applications découvertes, les besoins sur un nœud ne peuvent pas être assurés. Dans ce cas, pour éviter tout blocage définitif, seulement une partie des applications sera déployée.

Rapatriement

La phase suivante consiste à choisir une configuration valide pour chaque besoin identifié et à effectuer le transfert des implantations depuis les fournisseurs d'applications vers le nœud de déploiement.

Dans la solution proposée, l'algorithme essaye de minimiser le nombre de rapatriement à faire en favorisant les applications composites multimédia par rapport à celles des autres classes, et en favorisant les applications composites monomédia par rapport à celles de la classe monolithique. En effet, on préfère déployer une seule application qui traite plusieurs types de données que de déployer une application par type de donnée. De cette façon, on minimise les opérations de rapatriement. Par ailleurs, si une implantation est la solution pour plusieurs besoins identifiés, elle ne sera rapatriée qu'une seule fois.

L'algorithme ordonne les configurations en attribuant un poids à chaque configuration. Des poids sont aussi associés à chaque composant comme suit : la valeur 3 est attribuée au composant composite multimédia, la valeur 2 est attribuée au composant composite monomédia et enfin la valeur 1 est attribuée au composant monolithique

Le calcul du poids d'une configuration se fait comme suit :

```
1. Pour chaque configuration de déploiement valide faire
2.     Poids = 0
3.     Pour chaque implantation Ij de la configuration faire
4.         Poids = Poids + Poids de Ij
5.     Fin
6.     Nbre = nombre d'implantations dans la configuration
7.     Poids = Poids + suffix(Nbre)
8. Fin
```

```
1. Fonction suffix (Nbre) :
2.     j = 0
3.     i = nbre total d'implantations
4.     Tant que i > Nbre faire
5.         j = j + i * 3
6.         i - -
7.     Fin
8. retourne j
```

De cette façon, on favorise les groupes comprenant des applications composites multimédia par rapport à ceux comprenant des applications composites monomédia et monolithiques.

N°	Poids	Configuration	n2	n3	n4
1	**18**	I_1	✕	✕	✕
4	**14**	I_1, I_2	✕	✕	✕
5	**13**	I_1, I_3	✕	✕	✕
7	**6**	I_1, I_2, I_3	✕	✕	✕

Tableau 3.7. Réorganisation des configurations de déploiement

Instanciation des applications déployées

La dernière étape de notre algorithme consiste à instancier les applications déjà rapatriées et à les démarrer. Le nombre d'instanciation est déterminé par la capacité de l'application à supporter plusieurs flux de données d'un même type ou non. Une application productrice (resp. consommatrice) multiple ne sera instanciée qu'une seule fois. Une application productrice (resp. consommatrice) simple sera instanciée une fois par flux, soit N fois pour N flux. Cette information est disponible dans le descripteur de l'application (champ Diffusion de la figure 3.5).

Revenons à notre exemple et supposons que l'implantation I_1 a été rapatriée sur le terminal n1. Si I_1 permet de traiter plusieurs flux du même type alors elle n'est instanciée qu'une seule fois (figure 3.16, partie gauche). Sinon, elle est instanciée trois fois afin de servir les nœuds n2, n3 et n4 (figure 3.16, partie de droite).

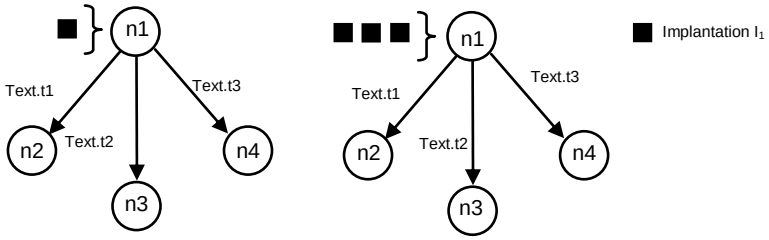


Figure 3.16. Instanciation pour un même besoin

Dans les étapes précédentes, l'algorithme de déploiement traite chaque besoin indépendamment des autres besoins jusqu'à la phase de rapatriement. Cependant, les configurations valides pour chaque besoin peuvent contenir des implantations en commun. Dans ce cas, le rapatriement n'est fait qu'une seule fois. Ainsi, si une implantation I figure dans deux configurations valides pour des besoins différents et si elle permet aussi de traiter plusieurs flux, alors elle ne doit pas être instanciée plusieurs fois mais seulement une fois (figure 3.17).

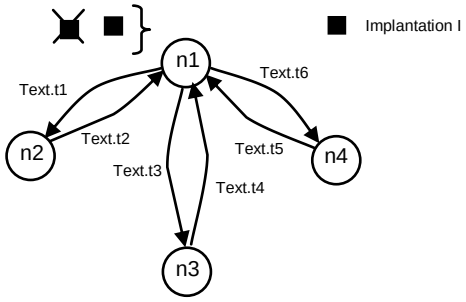


Figure 3.17. Instanciation pour des besoins différents

3.4.3 Déploiement ultérieur (ou reconfiguration)

Introduction

Le déploiement ultérieur soulève plus de défi que le déploiement initial. Il est plus difficile à traiter. Si chaque nœud exécute les procédures de déploiement, l'étape de vérification de l'interopérabilité posera problème puisque chacun vérifiera l'interopérabilité avec ses voisins qui à leur tour veulent vérifier l'interopérabilité. L'entrelacement du déroulement concurrent de la phase de vérification d'interopérabilité cause une instabilité et une oscillation improductive de l'algorithme proposé. En effet, chaque nœud dispose des implantations candidates et veut extraire parmi elles celles qui sont interopérables par rapport à ses voisins. Mais puisque les voisins disposent eux aussi des implantations candidates et veulent aussi extraire celles qui sont interopérables alors il sera impossible d'atteindre un état cohérent où les implantations à déployer seront interopérables conformément à la nouvelle instance du modèle de session. La solution que nous avons retenue se base sur un mécanisme de jeton pour ne faire le déploiement ultérieur que sur un seul nœud en même temps.

L’algorithme se fait donc en deux phases :

- **Phase 1**. Sur chaque nœud, l’algorithme accepte la nouvelle instance du modèle de session. En le parcourant et en faisant des comparaisons avec les besoins identifiés lors de la phase précédente de déploiement ou de reconfiguration, l’algorithme détermine la liste des nœuds qui sont affectés par l’évolution de la session. Afin d’appliquer le mécanisme de jeton, cette liste doit être réorganisée de façon à avoir le même ordre chez tous les nœuds impliqués dans la reconfiguration. Cette étape est exécutée par les nœuds simultanément.

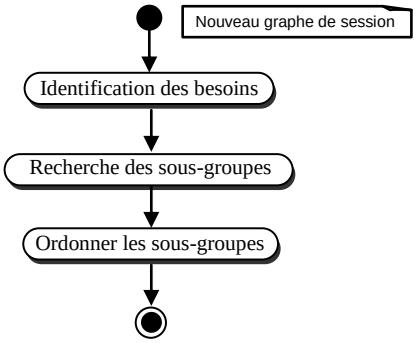


Figure 3.18. Etapes de la première phase du déploiement ultérieur

- Dans cette deuxième étape, seul le nœud ayant le jeton va pouvoir exécuter les opérations de la reconfiguration qui se basent sur les étapes du déploiement initial (figure 3.19). Une fois qu’il a terminé le déploiement, il passe le jeton au nœud suivant dans la liste. Ce dernier effectue le même traitement, et ainsi de suite jusqu’au dernier nœud concerné. De cette façon, tous les nœuds impliqués dans la reconfiguration exécutent les opérations du déploiement ultérieur sans conflit.

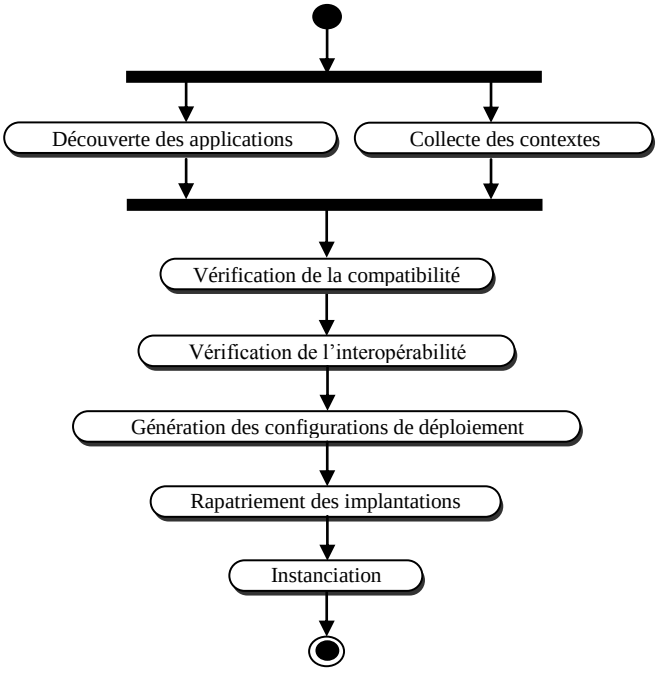


Figure 3.19. Etapes de la deuxième phase du déploiement ultérieur

Etapes de la reconfiguration

Lorsque la structure de la session change dans le temps, chaque nœud reçoit la nouvelle instance du modèle de session. Ainsi, l'algorithme s'exécutant sur le nœud, accepte alors ce nouveau graphe et commence la première phase de la reconfiguration.

Détermination des sous-ensembles pour la reconfiguration

En parcourant le nouveau graphe, l'algorithme détermine les nœuds qui nécessitent une reconfiguration. Ce graphe peut contenir plusieurs sous-ensembles impliqués dans la reconfiguration. Le parcours du graphe commence à partir du nœud qui exécute l'algorithme. De façon itérative, pour chacun des nœuds considérés, l'algorithme commence d'abord par identifier les besoins du nœud comme décrit précédemment. Puis, il compare les nouveaux besoins avec ceux du déploiement (initial ou ultérieur) précédent et décide si le nœud doit effectuer une reconfiguration ou pas.

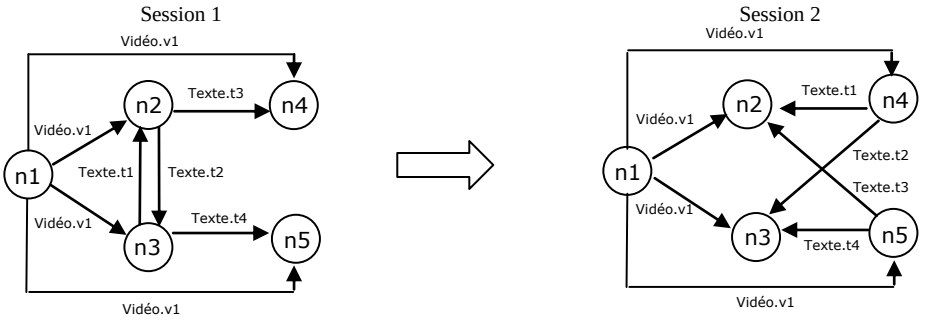


Figure 3.20. Evolution de la session

La figure 3.20 montre l'évolution de la structure de la session. Le nouvel état signifie que le nœud n4 (resp. n5) produit deux flux textuels, un vers n2 et l'autre vers n3. Ainsi, les nœuds affectés par cette évolution sont : {n2, n3, n4, n5}. Le processus de reconfiguration sera exécuté seulement sur ces nœuds. Par contre, le nœud n1 reste inactif.

Réorganisation du sous-ensemble des nœuds

A ce stade, les nœuds d'un même sous-ensemble auront chacun la même liste des nœuds impliqués dans la reconfiguration mais pas dans le même ordre. Afin d'attribuer correctement le jeton et exécuter la suite des opérations de déploiement, cet ensemble doit être ordonné. On utilise pour cela le critère de classement qui suit et qui permet d'obtenir la même liste ordonnée chez tous les nœuds, à partir d'ensembles ordonnés différemment.

Le nœud ayant le nombre de besoins en production plus élevé que les autres, sera placé en haut de la liste. Si deux nœuds ont le même nombre de besoins en production, alors celui qui émet le plus de flux de données en production, devancera l'autre. Dans le cas où c'est le même nombre de flux de données en production pour les deux nœuds, alors celui qui possède le nombre de besoins en consommation le moins élevé, devancera l'autre. Si ces deux nœuds ont le même nombre de besoins en consommation, alors celui qui reçoit le moins de flux de données en consommation, devancera l'autre. Enfin, dans le cas où les deux nœuds ont exactement les mêmes nouveaux besoins, l'ordre est donné par les identificateurs des nœuds.

Dans l'exemple précédent (figure 3.20), les nœuds n4 et n5 ont le nombre des nouveaux besoins en production {Texte, production} plus élevé. Ils seront classés avant les nœuds n2 et n3. L'ordre du sous-groupe de reconfiguration devient :

Avant réorganisation	Après réorganisation
{n2, n3, n4, n5}	{n4, n5, n2, n3}

Tableau 3.8. Réorganisation des nœuds

Le nœud qui détient le jeton exécute le reste des opérations de déploiement incluant la collecte des contextes des nœuds voisins, la découverte des applications, ainsi que la génération des configurations valides et l'instanciation des implantations après leur rapatriement.

Nous avons identifié tous les cas possibles de la reconfiguration.

Cas 1 : Ajout d'un nouveau besoin {type de données, sens de communication}

Ce cas est le plus simple à traiter. Les nœuds concernés vont produire ou consommer un nouveau type de données (figure 3.21). Après avoir identifié le nouveau besoin, le nœud concerné, exécute le reste des opérations de reconfiguration comme décrit précédemment.

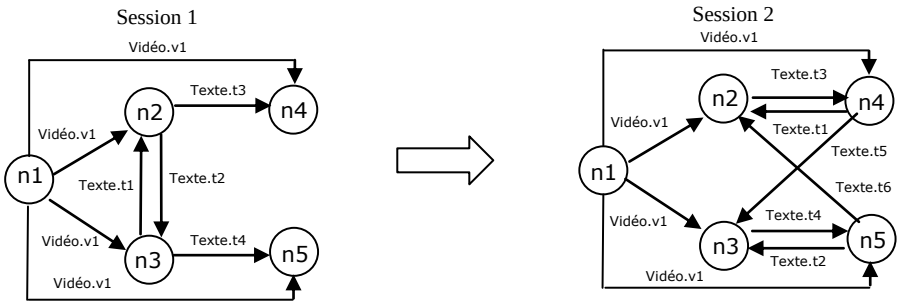


Figure 3.21. Ajout d’un nouveau besoin

Conformément au nouvel état de la session, les nœuds n4 et n5 possèdent un nouveau besoin à savoir {Texte, production}.

Cas 2 : Suppression d'un besoin

Ce cas correspond à une évolution de la session dans laquelle le nœud ne produit plus ou ne consomme plus des flux d'un type de données. Au niveau du modèle de session, cette évolution implique la suppression de certains arcs entre le nœud concerné et ses voisins (figure 3.22).

Pour chaque besoin supprimé, l'algorithme détermine parmi les implantations déployées, celles qui couvrent ce besoin. Dans le cas où une implantation est impliquée dans d’autres besoins, alors elle ne doit plus produire des flux de données vers les nœuds liés au besoin supprimé. Ainsi, si l’implantation appartient à la catégorie composite multimédia, alors elle doit diminuer le nombre de flux produit. Si l’implantation appartient à la catégorie monomédia, alors le nombre d’instance doit être diminué. Dans le cas où l’implantation n’est pas impliquée dans d’autres besoins, alors elle est gardée en cache pour d'autres utilisations.

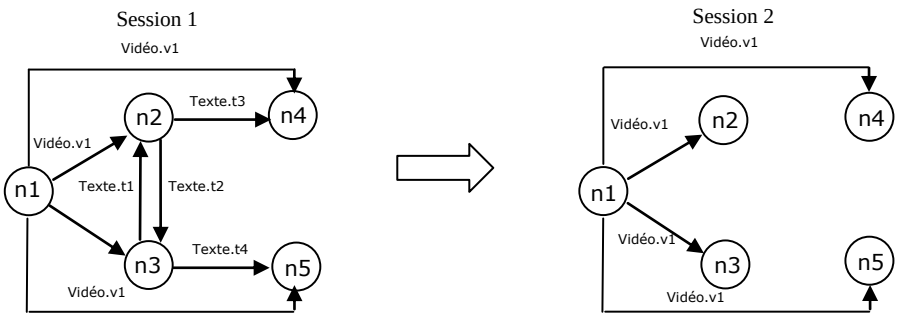


Figure 3.22. Suppression d’un besoin

Sur la figure 3.22, conformément au nouvel état de la session, les nœuds n2 et n3 ne traitent plus (en production et en consommation) les flux de données de type Texte. Les nœuds n4 et n5 ne consomment plus des flux de données de type Texte. {n2, n3, n4, n5} verront les instances des applications qui manipulent du texte, supprimées.

Cas 3 : Etablissement et rupture des communications en conservant les types des flux

Cette évolution consiste à arrêter les communications actuelles et à établir de nouvelles communications avec d'autres nœuds de la session. Le besoin est gardé mais la liste des nœuds voisins a été modifiée. Au niveau du modèle de session, cela implique la suppression de tous les arcs actuels (restreints au besoin à traiter) entre le nœud et ses voisins et l'ajout de nouveaux arcs vers d'autres nœuds (figure 3.23).

L'algorithme vérifie si les implantations déployées, répondent au nouveau cas. Il vérifie particulièrement si ces implantations sont interopérables avec les implantations déployées sur ces nouveaux nœuds voisins. Dans le cas où les implantations ne couvrent pas le besoin, l'algorithme effectue une découverte des applications candidates et poursuit le reste des opérations de déploiement.

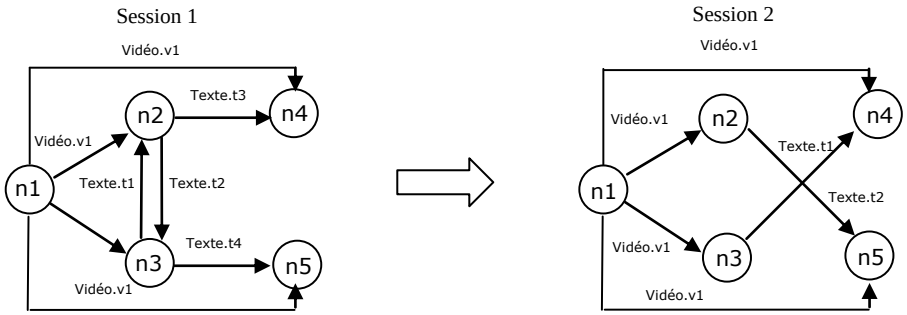


Figure 3.23. Etablissement de nouveaux liens et rupture des liens actuels

Le nœud n4 (resp. n5) ne consomme plus le flux textuel provenant du nœud n2 (resp. n3) mais plutôt celui provenant du nœud n3 (resp. n2).

Cas 4 : Rupture de quelques communications actuelles sans l'établissement de nouvelles communications

Ce cas est facile à traiter puisqu'il signifie que, pour un type de données particulier, le nombre de voisins diminue. Le nœud arrête donc la production ou la consommation de seulement quelques flux de données en gardant la communication avec les autres nœuds. Au niveau du modèle de la session, seulement quelques arcs touchés par l'évolution de la session vont être éliminés (figure 3.24). L'algorithme vérifie s'il n'y a pas des implantations à effacer puisque certaines communications avec des nœuds voisins ont été arrêtées.

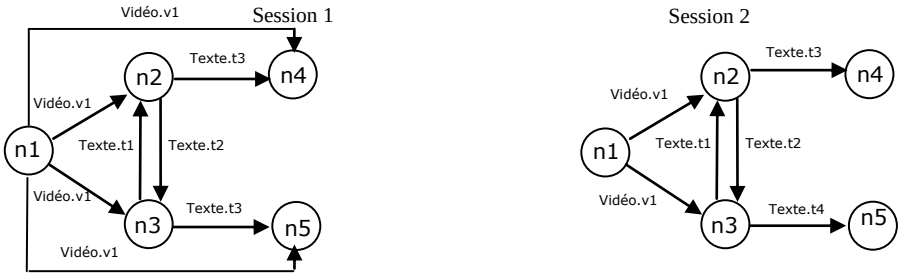


Figure 3.24. Rupture de quelques liens actuels sans l'établissement de nouveaux liens

Le nœud n1 ne produit plus un flux vidéo vers n4 et n5. Par contre, il garde toujours les liens vers les nœuds n2 et n3.

Cas 5 : Etablissement de nouvelles communications avec d'autres nœuds en gardant les communications actuelles

Pour un type de données particulier, le nombre des voisins augmente. Cela signifie que le nœud va pouvoir produire ou consommer plus de flux de données tout en gardant la communication avec les autres nœuds. Au niveau du modèle de la session, cette évolution se manifeste par l'ajout de quelques arcs avec des nœuds en gardant les arcs actuels (figure 3.25).

L'algorithme commence d'abord par récupérer les contextes des nouveaux voisins pour ensuite vérifier si les implantations actuelles couvrent la nouvelle situation. Dans le cas où les implantations actuelles ne répondent pas au nouveau besoin, l'algorithme découvre d'autres applications. Les implantations découvertes et qui sont compatibles localement, seront ajoutées à celles déjà déployées pour ensuite générer des configurations de déploiement valides.

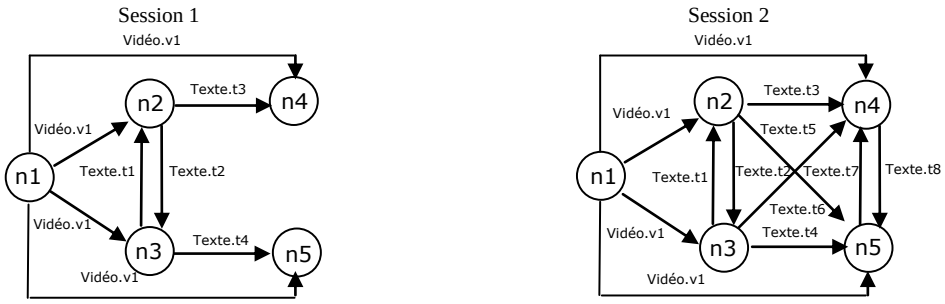


Figure 3.25. Etablissement de nouveaux liens en gardant les liens actuels

Cas 6 : Etablissement et rupture de communications avec des types de flux différents

Ce cas est plus général. Les nœuds concernés voient les sens de communication modifiés, mais aussi les types de flux manipulés sont changés. Pour un besoin particulier, le nœud arrête la production ou la consommation de quelques flux de données et établit de nouvelles communications avec d'autres nouveaux nœuds voisins. Dans l'instance du modèle de session, ce changement se reflète par l'ajout et la suppression des arcs entre le nœud et ses voisins comme décrit dans l'exemple de la figure 3.26.

Après avoir récupéré le contexte des nouveaux voisins, l'algorithme vérifie si les implantations actuelles couvrent la nouvelle situation. Si ce n'est pas le cas, l'algorithme découvre d'autres applications qui seront ajoutées à la liste actuelle et poursuit le reste des opérations du déploiement. Si des implantations déployées ne sont plus utiles, l'algorithme vérifie d'abord si elles ne sont pas impliquées dans d'autres besoins. Dans le cas contraire, il les garde en cache pour d'autres utilisations.

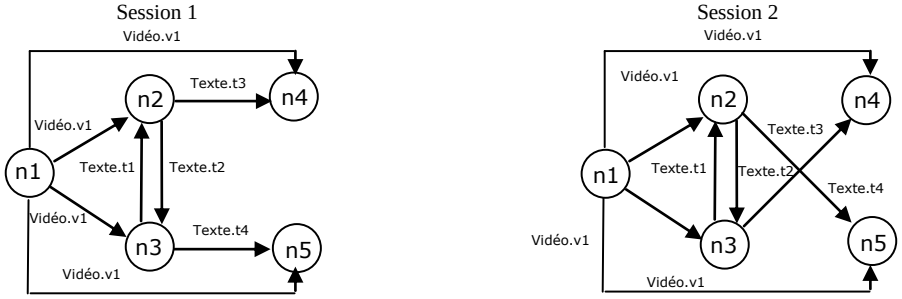


Figure 3.26. Etablissement de nouveaux liens et rupture de quelques liens actuels

Le nœud n2 (resp. n3) ne produit plus un flux de données de type Texte vers le nœud n4 (resp. n5). Par contre, il produit un nouveau flux vers le nœud n5 (resp. n4).

3.5 Conclusion

Nous avons présenté dans ce chapitre l'algorithme de déploiement et de reconfiguration. Nous avons surtout insisté sur les différentes étapes à suivre pour arriver à une configuration de déploiement valide qui couvre le rôle du participant à la session et qui respecte les contraintes de compatibilité et d'interopérabilité. Nous avons aussi expliqué les différents modèles de session, d'applications et de nœuds. Ces modèles sont utilisés par l'algorithme pour choisir les applications appropriées.

	JWS	Software Dock	ORYA	CODEWAN	AMPROS	Planit	SDC
Stratégie	Pull	Push-Pull	Push	Push-Pull	Push	---	Pull
Catégorie	Statique	Statique	Statique	Dynamique	Dynamique	Dynamique	Dynamique
Types	Initial	Initial	Initial	Initial	Initial-Ultérieur	Initial-Ultérieur	Initial-Ultérieur
Contrôle	Centralisé	Centralisé	Centralisé	Décentralisé	Centralisé	Centralisé	Décentralisé
Sensibilité au contexte	---	Passif	Passif	----	Passif	Passif	Passif

Tableau 3.9. Position de notre algorithme SDC de déploiement et de reconfiguration

La dernière colonne du tableau 3.9 donne les principales caractéristiques de notre contribution, appelée Système de Déploiement Collaboratif (SDC), par rapport aux principaux travaux identifiés et analysés dans notre état de l'art. Nous voyons que notre approche se situe dans un contexte dynamique et décentralisé, dans lequel chaque nœud opère de façon autonome en respectant les contraintes locales et globales.

Le chapitre suivant est consacré à décrire la plate-forme de nous avons développée pour supporter l'algorithme de déploiement et de reconfiguration. Il explique son architecture Pair-à-Pair ainsi que les interfaces et les structures de données mises en œuvre.

Chapitre 4 Conception de la plate-forme de déploiement et de reconfiguration

4.1 Introduction

L'objectif de ce chapitre est de combiner les deux technologies, celle du déploiement sensible au contexte et celle des réseaux Pair-à-Pair, pour offrir aux développeurs des services leur permettant de réaliser des applicatifs plus complexes basés sur le déploiement adaptatif. Nous proposons donc cette plate-forme qui implante notre algorithme de déploiement et de reconfiguration décrit plus haut. Les principaux buts sont :

- d'une part, de fournir une abstraction de la complexité des réseaux Pair-à-Pair et des opérations de déploiement et de reconfiguration ;
- et d'autre part, d'offrir un ensemble de services communs aux applications de déploiement sensible au contexte.

Nous présentons en premier lieu les fonctionnalités requises pour réaliser le déploiement et la reconfiguration. Ensuite, nous détaillons l'architecture en décrivant les modules du système et les principales méthodes offertes.

4.2 Fonctionnalités requises pour le déploiement

Afin de déterminer les fonctionnalités nécessaires pour le déploiement sensible au contexte en mode Pair-à-Pair, nous avons tout d'abord identifié toutes les fonctionnalités requises dans un scénario de déploiement qui illustre la connexion d'un participant dans la session (figure 4.1). Nous avons ensuite factorisé ces fonctionnalités dans des opérations génériques regroupées en quatre grandes familles : (1) accès au réseau P2P, (2) interaction locale, (3) interaction distribuée et (4) contrôle du déploiement.

4.2.1 Accès au réseau P2P

Fonctionnalité générique

La première fonctionnalité concerne les activités communes de toute application P2P, à l'exception des fonctionnalités d'échange de messages. On retrouve ainsi les opérations de recherche de groupe de pairs en ligne, de connexion ou de déconnexion. Un groupe permet aux pairs de se constituer en communautés virtuelles. En dépit du contexte d'utilisation, des équipements utilisés et des réseaux les supportant qui sont très variés, les opérations d'accès au réseau Pair-à-Pair sont les mêmes. C'est pourquoi, nous les regroupons dans un module générique d'*Accès au réseau P2P*.

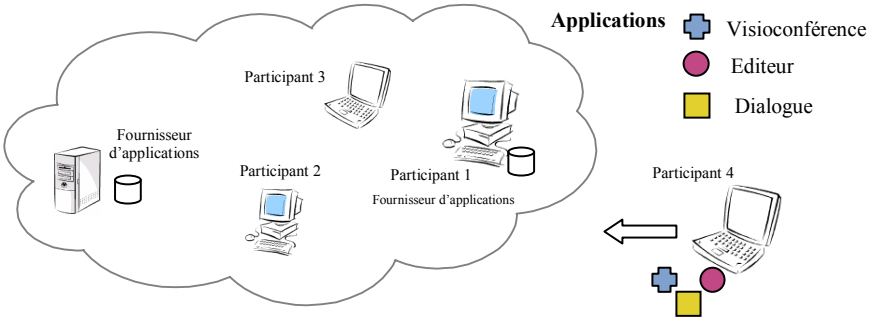


Figure 4.1. Scénario d'entrée d'un nouveau participant

Utilisation dans le cadre du déploiement

La session collaborative est formée par des participants. Les membres de la session et leurs équipements sont les nœuds formant le groupe de pairs. Un nœud peut ainsi rejoindre la session en cours (ou démarrer une nouvelle session) et être dans le même espace virtuel que les autres nœuds. L'utilisation du module *Accès au réseau P2P* sert à retrouver (ou créer) la session et à s'y connecter.

4.2.2 Interaction locale

L'interaction locale fait référence aux échanges entre le nœud et un répertoire du système de fichiers. Ce répertoire est un dépositaire local qui sert à stocker des informations telles que les différents descripteurs.

Fonctionnalité générique

Le descripteur d'application, le descripteur du contexte d'un nœud et le descripteur décrivant les applications instanciées sur un nœud voisin après déploiement sont représentés sous la forme d'un document structuré. Ce document est composé d'une arborescence d'éléments simples qui sont constitués d'une clé et d'une valeur, ou d'éléments composites, contenant d'autres éléments simples ou composites. Nous définissons la notion de contenu générique pour représenter le descripteur d'une application ou autre. Les opérations d'accès au dépositaire de contenus, sont les mêmes, quel que soit le type du contenu. On définit donc un module générique *Gestion de contenu* pour supporter ces opérations (figure 4.2).

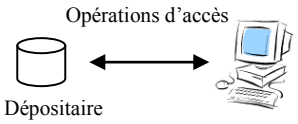


Figure 4.2. Interaction locale

Utilisation dans le cadre du déploiement

L'utilisation intervient dans quatre types d'accès nécessaires :

- Chaque nœud possède un descripteur de contexte enregistré dans un dépositaire local. Cette interaction a pour but de récupérer ce descripteur. Pour cela, le nœud interroge localement ce répertoire ;

- Une application est décrite par un descripteur qui est enregistré dans un dépositaire local. Le nœud accède à ce répertoire pour récupérer le descripteur ou bien pour l'enregistrer ;

- Sur les nœuds voisins, après la phase de déploiement, un descripteur décrivant les applications déployées est créé. Il est enregistré dans un dépositaire local. Pour pouvoir publier et envoyer ces descripteurs aux nœuds demandeurs, le nœud voisin doit donc interagir avec ce dépositaire.

4.2.3 Interaction distribuée

L'interaction distribuée se fait entre les nœuds qui représentent les pairs du réseau pour s'échanger soit des contenus, soit les entités qui représentent ces contenus.

Fonctionnalité générique

On distingue deux comportements génériques celui de *fournisseur réactif* et celui de *demandeur proactif*.

Le fournisseur réactif est responsable :

(1) d'informer les autres pairs de l'existence des contenus. Pour cela, il les déclare en publiant une annonce pour chaque contenu. Cette annonce contient des éléments décrivant le contenu comme l'identifiant, le nom, le type ou la classe, les mots-clés. L'annonce contient également un élément spécifiant l'accès au nœud qui la publie. Ceci est utile pour qu'un autre nœud puisse contacter le nœud fournisseur. Selon la topologie Pair-à-Pair choisie, une annonce peut être publiée localement, c'est-à-dire qu'elle est stocké sur le nœud et devient accessible pour l'extérieur. Elle peut aussi être publiée chez un ou plusieurs nœuds qui la référencent.

(2) de répondre aux requêtes de rapatriement des nœuds initiateurs. Le fournisseur se met en attente des connexions des autres nœuds. Lorsqu'une demande lui parvient, il l'analyse pour savoir s'il s'agit d'une demande de rapatriement d'un contenu ou de l'entité représentée par le contenu. Ensuite, il répond au nœud demandeur en envoyant l'objet demandé s'il le trouve, sinon il envoie un message d'erreur.

Le demandeur proactif initialise :

(1) Les opérations de découverte. Les annonces émises par les nœuds lors de la publication des contenus sont recherchées par les autres nœuds. La découverte permet donc à un nœud de savoir que des contenus susceptibles de l'intéresser, sont disponibles sur un lieu précis. Après la découverte, le nœud demandeur reçoit des annonces qui répondent à sa requête. Il peut ensuite choisir de les conserver ou non ; cela dépend de la pertinence de l'annonce par rapport à ses besoins.

(2) Les opérations de rapatriement. A partir de l'annonce découverte, le nœud demandeur extrait l'adresse du fournisseur et établit une connexion directe avec lui pour le rapatriement du contenu référencé par l'annonce. Par la suite, il pourra lancer le rapatriement de l'entité représentée par le contenu. Cette phase de rapatriement est réalisée de la même façon pour des objets différents (contenu ou l'entité représentée par le contenu), sauf que le traitement après réception de l'objet demandé n'est pas le même. Grâce au concept de *listener*, il est possible de personnaliser ce traitement sans pouvoir réécrire le code de la classe qui reçoit le message. Lorsqu'un événement surviendra, tel que la réception d'un message, la classe qui devrait le traiter fait appel à la méthode de la classe personnalisée qui implante l'interface *listener* et définit donc le comportement personnalisé (c'est-à-dire les instructions personnalisées à exécuter).

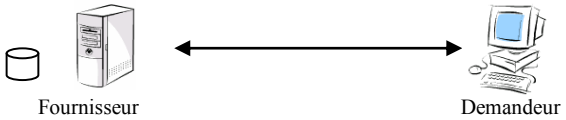


Figure 4.3. Interaction distribuée

Utilisation dans le cadre du déploiement

Chaque nœud peut supporter un ou plusieurs rôles distincts :

- Un nœud peut être considéré comme étant *fournisseur réactif* de descripteur d'application s'il dispose de ce document et il est en mesure de l'émettre après une demande explicite.

- Un nœud est susceptible d'être *demandeur proactif* de descripteur d'application. Il émet une requête vers le fournisseur.

- Un nœud peut aussi adopter un rôle *fournisseur réactif* d'archive contenant les implantations. Il répond aux demandes de rapatriement.

- Un nœud est capable d'avoir un rôle *demandeur* d'application et se comporter en mode *proactif* s'il envoie une requête de rapatriement.

- Le nœud peut avoir un rôle *fournisseur réactif* de descripteur des applications déployées pour la session en cours. Il répond aux demandes qui lui sont faites pour ce document.

- Enfin, le rôle *demandeur proactif* de descripteur des applications déployées consiste à demander et à recevoir un document décrivant les applications présentes sur un nœud voisin.

4.2.4 Contrôle du déploiement

Cette fonctionnalité permet de conduire les opérations du déploiement comme le fait de donner les ordres pour récupérer les différents descripteurs identifiés auparavant ou de procéder au filtrage des implantations. La reconfiguration suite à l'évolution de la session est

aussi effectuée à ce niveau. Le contrôle du déploiement a besoin des modules génériques précédents. Il se place donc dans une couche plus haute. Pour réaliser les opérations de déploiement, les participants doivent suivre les étapes suivantes :

(1) La première étape concerne la description des ressources requises par les applications et des ressources fournies par les environnements d'exécution des participants. Cette étape fait appel au module *d'Accès au dépositaire de contenu*.

(2) La deuxième étape, qui comprend les opérations pour qu'un participant se connecte à la session et soit en relation avec les acteurs du déploiement, fait appel aux fonctionnalités du module d*'Accès au réseau P2P.*

(3) La troisième étape, réalisée par chacun des participants, concerne la vérification de la compatibilité des ressources requises par les applications trouvées avec celles fournies par l'environnement d'exécution. Cette phase fait tout d'abord appel aux modules *demandeur proactif* et *fournisseur réactif* pour récupérer les descripteurs des applications disponibles, puis utilise le module *d'accès au dépositaire de contenu* pour connaître l'environnement d'exécution.

(4) La quatrième étape, faite aussi par chacun des participants, porte sur l'interopérabilité entre voisins des applications trouvées. Cette phase fait aussi appel aux modules *demandeur proactif* et *fournisseur réactif* pour récupérer les descripteurs des applications déployées sur les voisins.

(5) La cinquième étape, faite par chaque participant, concerne le rapatriement et l'instanciation des applications choisies. Cette phase fait une nouvelle fois appel aux modules *demandeur proactif* et *fournisseur réactif* pour récupérer les applications à instancier.

4.3 Architecture

En partant des fonctionnalités précédentes, nous avons fait la séparation entre un domaine générique caractérisé par des structures de données et des comportements uniformes, et un domaine orienté déploiement sensible au contexte. Cette séparation nous donne l'architecture en couche suivante (figure 4.4) :

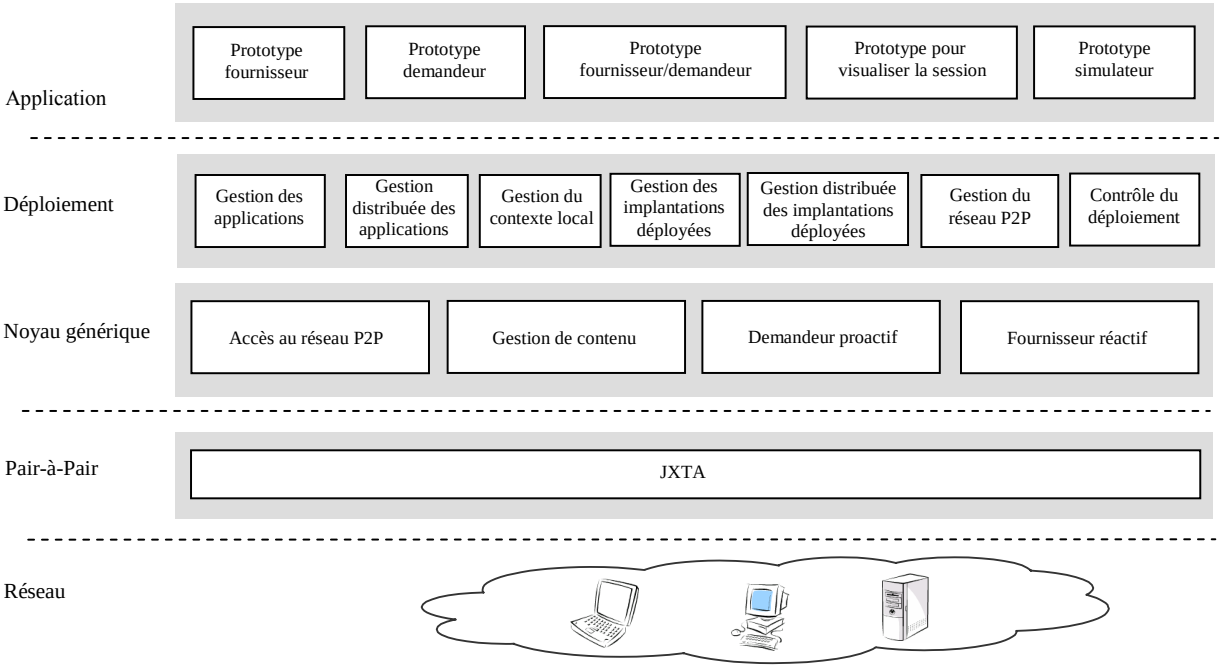


Figure 4.4. Architecture de la plate-forme de déploiement et de reconfiguration

Au cœur de cette architecture, on trouve des fonctions essentielles telles que la gestion des groupes et des communications entre les membres d'une même communauté de nœuds. Au dessus, ces fonctionnalités de base peuvent servir à créer des services de déploiement sensible au contexte dans les sessions collaboratives. Au plus haut niveau, les développeurs pourront développer des applications à valeur ajoutée se basant sur le déploiement adaptatif. Les différentes couches de cette architecture sont :

- La couche réseau est constituée des équipements informatiques utilisés pour se connecter à la session. Il est fréquent que des réseaux filaires et sans fils coexistent dans une même organisation.

- La couche Pair-à-Pair a pour but d'abstraire la couche réseau physique. Elle fournit une infrastructure unifiée pour le développement d'applications et de services P2P.

- La couche noyau générique permet de masquer la complexité de la couche Pair-à-Pair en offrant un ensemble de services communs aux applications P2P de partage de contenu.

- La couche déploiement utilise les services offerts par le niveau noyau générique pour effectuer les opérations relatives au déploiement sensible au contexte.

- La couche application représente le domaine d'utilisation. Les développeurs utilisent directement les services du niveau déploiement pour construire des applications de déploiement selon leur besoin.

- L'organisation en couche de notre plate-forme renforce sa possibilité d'adaptation à d'autres services de niveau supérieur. Cette flexibilité offerte par l'architecture accroît quelque peu le nombre de classes de notre système.

Avant de détailler l'architecture, il est important de noter que les éléments clés utilisés dans l'API de déploiement sont :

- D'une part, l'utilisation de la programmation évènementielle et les classes *listener* permettant ainsi de personnaliser les traitements sans toucher au code du niveau inférieur.

- Et d'autre part, la séparation entre les interfaces et les implantations permettant aux niveaux supérieurs de ne pas toucher leur code dans le cas de modifications au niveau inférieur comme par exemple le passage d'une version à une autre.

4.3.1 Couche Pair-à-Pair

La couche Pair-à-Pair offre une infrastructure contenant les éléments essentiels pour développer des applications P2P. Elle définit aussi la topologie adoptée pour le réseau Pair-à-Pair.

Nous utilisons JXTA [45] pour cette couche. Développé par Sun, JXTA est un projet ouvert dédié aux applications Pair-à-Pair. L'objectif principal est de fournir une plate-forme unifiée pour le développement d'applications et de services P2P. Ainsi, on trouve les pairs capables d'effectuer des traitements et de communiquer avec d'autres pairs. Ils sont regroupés dans des groupes de pairs. Le groupe de pairs est une notion abstraite pour former des communautés virtuelles en dépit de la répartition réelle des pairs. Les pairs communiquent entre eux via des canaux (ou pipes) qui sont composés de files de messages. Un canal est aussi une ressource JXTA et il possède un identificateur.

JXTA introduit aussi la notion d'annonce (*advertisement*), système de déclaration publique en XML. Une annonce est une description structurée d'une entité disponible sur le réseau P2P. Le protocole de découverte PDP (*Peer Discovery Protocol*) utilise ces annonces pour décrire et publier les ressources dont dispose un pair. Les nœuds, ou pairs, peuvent découvrir ces annonces à l'aide de ce protocole.

Avec le concept de Rendezvous pair, qui représente un nœud spécial, nous pouvons construire les trois topologies de réseaux P2P. La topologie centralisée s'obtient en n'instanciant qu'un seul Rendezvous pair dans le réseau. La topologie totalement décentralisée implique qu'il n'y a aucun Rendezvous pair. Enfin la topologie hiérarchique s'obtient avec plusieurs Rendezvous pairs. Ceci rend notre réalisation la plus générale possible en termes de topologies de réseaux P2P.

Par la suite, nous présentons les raisons qui ont motivé le choix de JXTA pour notre travail :

- L'algorithme de déploiement que nous avons proposé suit une architecture décentralisée. Il n'existe pas de serveur pour diriger les opérations de déploiement. Cependant, les nœuds interagissent et s'échangent des informations. Intuitivement, ces interactions suivent un modèle Pair-à-Pair. JXTA offre le moyen de raccorder des nœuds dans un groupe de pairs sans se soucier des contraintes de bas niveau. De plus, ces nœuds peuvent rejoindre et quitter le réseau volontairement.

- Le déploiement repose en particulier sur la recherche des applications qui couvrent les besoins des utilisateurs ainsi que sur la recherche des autres descripteurs comme celui décrivant la configuration sur un nœud voisin. Cela signifie que ces ressources doivent d'abord être décrites et publiées pour les autres nœuds. Le service de découverte fourni avec JXTA supporte la réalisation de ces opérations.

- Les nœuds jouent tour à tour le rôle de client, demandeur de ressources, et de serveur, fournisseur de ressources. Grâce aux *sockets* JXTA, il est possible d'implanter aisément les fonctions nécessaires pour le rapatriement des objets entre les nœuds.

4.3.2 Couche noyau générique

Cette couche ainsi que la couche suivante représentent notre contribution majeure. Elle est constituée de quatre modules dont nous détaillons les principales classes et interfaces. Ces modules, au travers d'un ensemble de méthodes génériques, abstraient la couche P2P sous-jacente et renforcent la modularité de la réalisation. Il est à noter que cette couche est

totalement indépendante du déploiement et peut être utilisée pour construire des applications de partage de contenu.

Module d'accès au réseau P2P

Ce module, formé de plusieurs interfaces et classes (figure 4.5), permet d'étendre les mécanismes de base offerts par la couche Pair-à-Pair et ainsi de faciliter le développement d'applications de plus haut niveau. Les principaux services proposés dans ce module sont des mécanismes de gestion de groupes, de recherche et de partage de ressources.

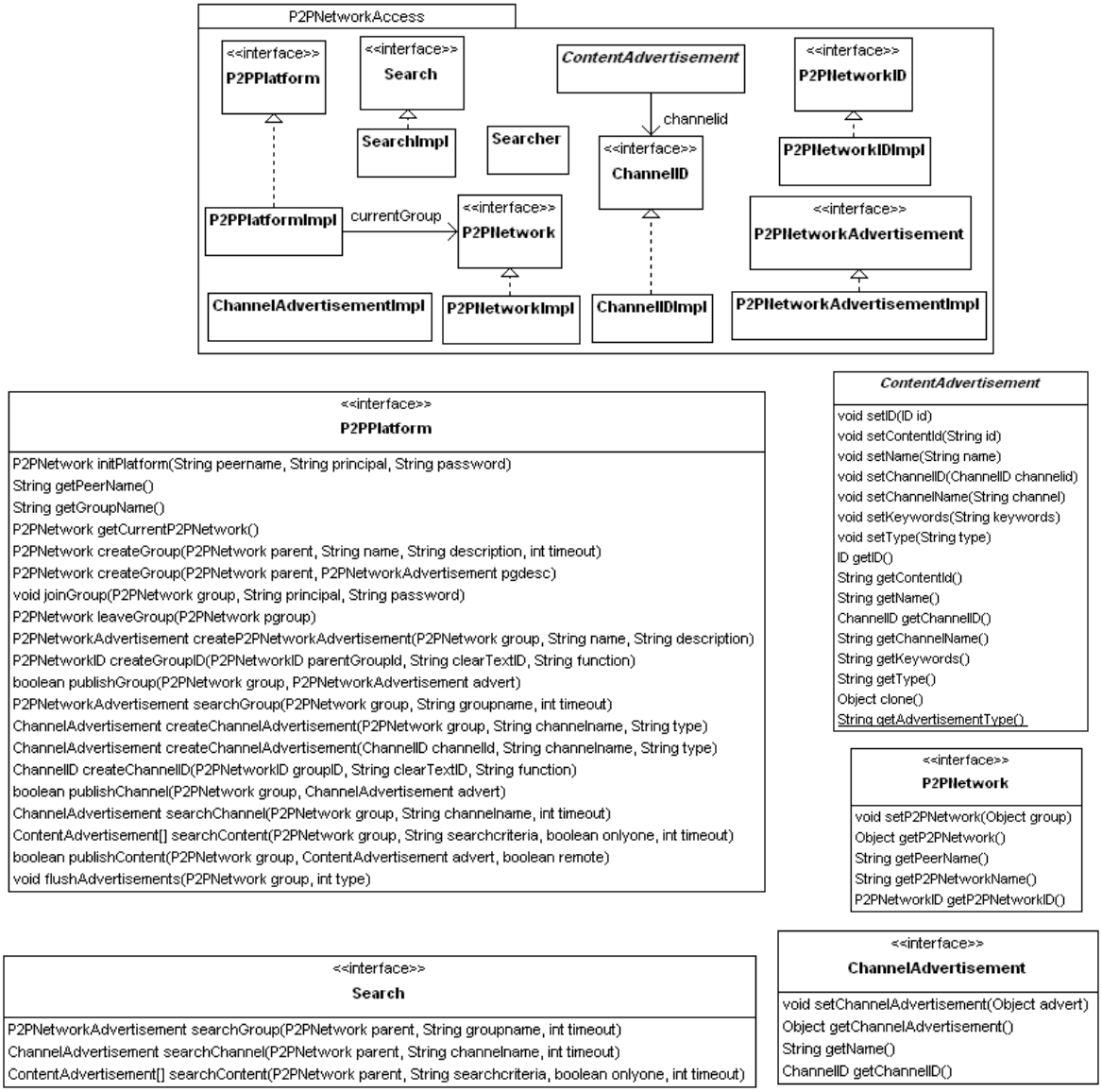


Figure 4.5. Module d'accès au réseau P2P

Les méthodes offertes par ce module sont classées logiquement en quatre groupes :

- **Gestion de groupe.** Les groupes permettent aux nœuds de se constituer en communautés ayant des intérêts communs. Un groupe est vu comme une ressource du réseau qu'un nœud peut choisir d'utiliser. Comme toutes les ressources, il possède un identifiant et est annoncé dans le réseau. Les fonctions illustrées dans le tableau 4.1 permettent à un nœud de s'insérer ou de se retirer du réseau Pair-à-Pair.

Méthodes	Description
P2PNetwork initPlatform (String peername, String user, String password)	Permet de démarrer l'application Pair-à-Pair. Cette fonction retourne l'instance du *NetPeerGroup.*
P2PNetwork createGroup (P2PNetwork parent, P2PNetworkAdvertisement advert)	Permet de créer un groupe de pairs. Il retourne l'instance du groupe.
void joinGroup (P2PNetwork group, String user, String password)	Permet de joindre un groupe de pairs existant.
P2PNetwork leaveGroup (P2PNetwork group)	Permet de quitter un groupe de pairs. Elle retourne l'instance du groupe que l'on vient de quitter.

Tableau 4.1Méthodes pour la gestion de groupe

- **Création d'annonce**. Les annonces (*advertisements*) décrivent les ressources disponibles dans le réseau Pair-à-Pair. Ces ressources peuvent être un groupe, un canal ou un contenu. Les classes *P2PNetworkAdvertisement, ChannelAdvertisement* et *ContentAdvertisement* sont instanciées avec les paramètres adéquats pour créer des annonces. Afin de ne pas créer une classe d'annonce par un type de contenu, nous avons choisi d'utiliser une seule annonce qui contient uniquement les informations communes, c'est-à-dire l'identificateur, le nom, le type, une liste de mots clés et l'adresse du nœud fournisseur publiant le contenu. Cette adresse est formée par l'identificateur du canal (*ChannelID*) et son nom (*ChannelName*). Puisque la structure d'un contenu varie d'un type de contenu à un autre, on a conçu une fonction générique *createContentAdvertisement()* qui se charge d'extraire les informations à encapsuler dans la classe *ContentAdvertisement*. Cette fonction sera spécialisée selon la classe de chaque contenu. Le tableau 4.2 liste l'ensemble des méthodes nécessaire à la création d'annonce.

Méthodes	Description
P2PNetworkID createGroupID (P2PNetworkID parentGroupId, String clearTextID, String function)	Crée l'identificateur du groupe de pairs. Cette fonction retourne l'instance de la classe *P2PnetworkID*.
P2PNetworkAdvertisement createGroupAdvertisement(P2PNetwork parent, String name, String description)	Génère l'annonce d'un groupe de pairs. Elle retourne l'instance de la classe *P2PnetworkAdvertisement.*
ChannelID createChannelID (P2PNetworkID groupID, String clearTextID, String function)	Crée l'identificateur d'un canal. Cette fonction retourne l'instance de la classe *ChannelID.*
ChannelAdvertisement createChannelAdvertisement(ChannelID channelId, String channelname, String type)	Génère l'annonce d'un canal et retourne l'instance de la classe *ChannelAdvertisement*

Tableau 4.2. Méthodes pour la création d'annonce

- **Publication**. Publier une annonce revient à la mettre dans un cache local que le fournisseur parcourt lorsqu'il reçoit une requête de découverte. Les méthodes du tableau 4.3 permettent la réalisation de cette opération.

Méthodes	Description
boolean publishGroup (P2PNetwork group, P2PNetworkAdvertisement advert)	Publie l'annonce du groupe de pairs. La fonction retourne *true* si la publication a réussi, *false* sinon.
boolean publishChannel (P2PNetwork group, ChannelAdvertisement advert)	Publie l'annonce d'un canal et retourne *true* si la publication a réussi, *false* sinon.
boolean publishContent (P2Pnetwork group, ContentAdvertisement advert, boolean remote)	Publie l'annonce d'un contenu et *true* si la publication a réussi, *false* sinon.
void flushAdvertisements (P2PNetwork group, int type)	Efface les annonces dans le cache ayant comme type groupe de pairs, canal ou contenu

Tableau 4.3. Méthodes pour la publication

- **Découverte**. Il est possible de réaliser la découverte grâce aux fonctions citées dans le tableau 4.4. Du fait que les communications dans le réseau P2P sont asynchrones et afin de ne pas attendre indéfiniment, la découverte est limitée par une période de temps passée en paramètre (*timeout*). De plus, afin de ne pas interrompre l'exécution du programme informatique, la découverte est effectuée dans un *thread* à part. Ceci a l'avantage de pouvoir lancer plusieurs opérations de découverte à la fois.

Méthodes	**Description**
P2PNetworkAdvertisement searchGroup (P2PNetwork group, String groupname, int timeout)	Recherche une annonce de groupe de pairs. Cette fonction retourne une instance de la classe *P2PNetworkAdvertisement* si elle l'a trouvé, *null* sinon.
ChannelAdvertisement searchChannel (P2PNetwork group, String channelname, int timeout)	Recherche une annonce d'un canal. La fonction retourne une instance de la classe *ChannelAdvertisement* si elle l'a trouvé, *null* sinon.
ContentAdvertisement[] searchContent (P2PNetwork group, String searchcriteria, boolean onlyone, int timeout)	Recherche les annonces des contenus et retourne un tableau des instances de la classe *ContentAdvertisement*.

Tableau 4.4. Méthodes pour la découverte

Module de gestion de contenu

Ce module offre une API ayant pour objectif d'assister les développeurs dans les opérations relatives à la gestion des contenus. Il permet en outre de représenter différentes entités, par exemple un fichier d'archive, sous un format facilement manipulable par un programme informatique. Ces entités sont décrites par des attributs regroupés dans une structure hiérarchique plus ou moins complexe selon le contexte cible. Ce concept hiérarchique provient du langage XML. Dans la suite, nous allons détailler l'API exportée par ce module :

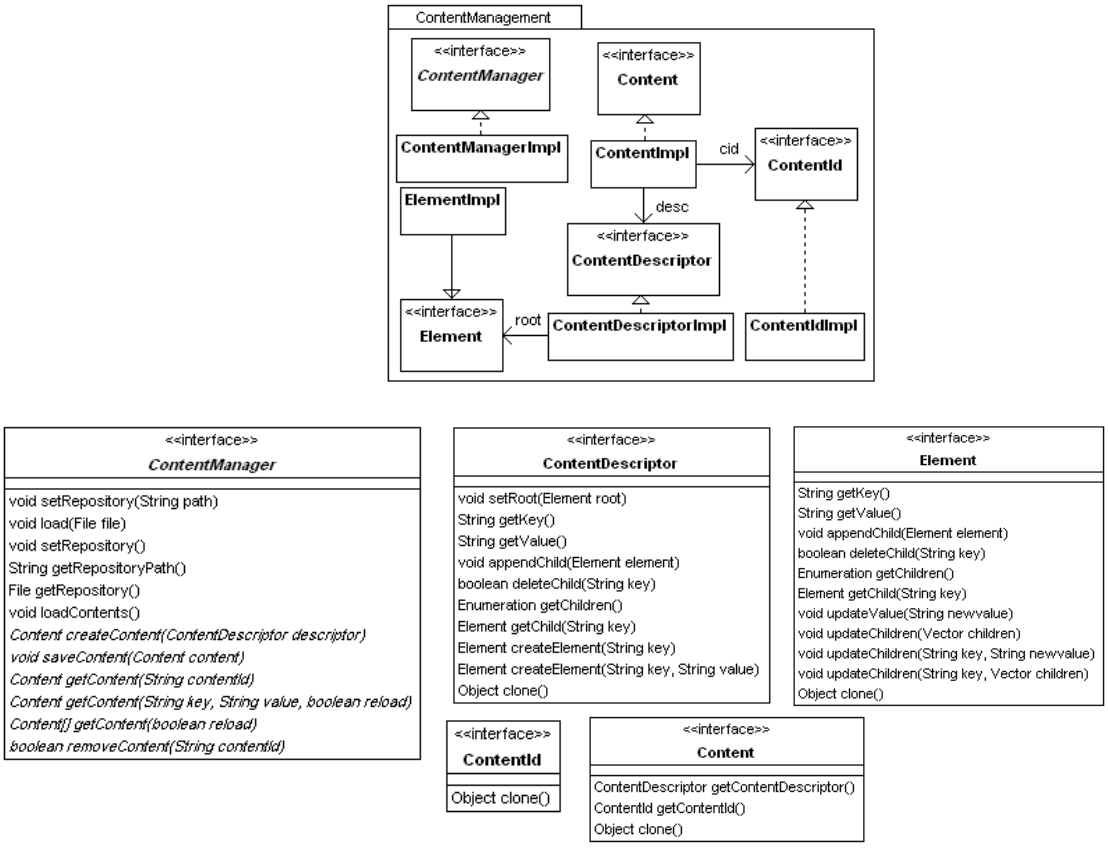


Figure 4.6. Module d'accès au dépositaire de contenu

- ***Content.*** L'interface *Content* représente le contenu qu'on veut manipuler. Elle possède deux attributs implantant les interfaces *ContentID* et *ContentDescriptor* que l'on peut récupérer avec les méthodes du tableau 4.5 :

Méthodes	**Description**
ContentDescriptor getContentDescriptor ()	Retourne l'instance du descripteur de contenu *ContentDescriptorImpl*.
ContentID getContentId ()	Retourne l'instance de l'identificateur de contenu *ContentIDImpl.*
Object clone ()	Retroune une copie de l'instance *ContentImpl*

Tableau 4.5. Méthodes offertes par l'interface *Content*

- ***ContentID*** permet d'attribuer un identificateur unique pour le contenu. Cet identificateur est généré en appliquant une fonction de hachage sur le descripteur de contenu passé en paramètre au constructeur de *ContentIDImpl.*

- ***ContentDescriptor.*** Les attributs décrivant un contenu sont enregistrés dans un objet *ContentDescriptor*. Celui ci admet un élément composé qui est la racine du descripteur, formé par une clé et un tableau d'éléments simples ou composés. La clé renseigne sur le type du contenu. Le tableau 4.6 donne les attributs du descripteur. *ContentDescriptor* et *Element* sont conjointement utilisés pour créer des descripteurs avec des degrés de complexité variés.

Méthodes	**Description**
void setRoot (Element root)	Permet d'initialiser l'élément racine (niveau zéro) du descripteur. Le paramètre root définit le type du descripteur. Il peut être aussi passé dans le constructeur de *ContentDescriptorImpl*.
Element createElement (String key)	Retourne l'instance d'un élément composite. Key est la clé de cet élément.
Element createElement (String key, String value)	Retourne l'instance d'un élément simple. Key est la clé de cet élément, *value* est sa valeur.
void appendChild (Element element)	Permet de rajouter un élément simple ou composite à la racine du descripteur
boolean deleteChild(String key)	Permet de supprimer l'élément au premier niveau du descripteur, ayant comme clé celle passée en paramètre. Elle retourne *true* si l'opération a réussi, *false* sinon.
Enumeration getChildren()	Retourne tous les éléments de la racine du descripteur.
Element getChild(String key)	Retourne l'élément du premier niveau du descripteur, ayant comme clé celle passée en paramètre
Object clone ()	Retourne une copie de l'instance *ContentDescriptorImpl*

Tableau 4.6. Méthodes offertes par l'interface *ContentDescriptor*

- ***Element.*** Cette interface (tableau 4.7) offre des méthodes pour créer des éléments simples ou composites. Un élément simple contient des attributs *clé* et *valeur* de type chaîne de caractères. Un élément composite contient un attribut *clé* de type chaîne de caractères et un attribut *fils (children)* de type tableau d'éléments.

Méthodes	**Description**
String getKey ()	Retourne la clé de l'élément
String getValue ()	Retourne la valeur de l'élément s'il s'agit d'un élément simple
void appendChild (Element element)	Permet de rajouter un sous-élément simple ou composite qui sera attaché à cet élément.
Enumeration getChildren ()	Retourne tous les sous-éléments attachés à cet élément.
Element getChild (String key)	Retourne le sous-élément ayant pour clé la valeur passée en paramètre.
void updateValue (String newvalue)	Met à jour l'attribut *value* de cet élément.
void updateChildren (String key, String newvalue)	Met à jour l'attribut *value* du sous-élément ayant pour clé le paramètre key.
void updateChildren(String key, Vector children)	Met à jour tous les sous-éléments du sous-élément ayant pour clé celle passée en paramètre

Tableau 4.7. Méthodes offertes par l'interface *Element*

- ***ContentManager.*** L'interface *ContentManager* se focalise sur la gestion des contenus en offrant des opérations facilitant l'accès au dépositaire de contenus :

Méthodes	**Description**
void setRepository (String path)	Permet de créer un dépositaire s'il n'existe pas, pour enregistrer les contenus
File getRepository ()	Retourne une instance de la classe *File* qui représente le dépositaire sous forme d'un répertoire du disque dur.
String getRepositoryPath ()	Retourne le chemin du dépositaire
void loadContents ()	Permet de charger tous les contenus du dépositaire dans la mémoire.
Content createContent (ContentDescriptor desc)	Crée un nouveau contenu ayant pour descripteur l'objet passé en paramètre. L'identificateur du contenu sera généré en se basant sur cet objet.
void saveContent (Content content)	Enregistre le contenu dans le dépositaire.
Content getContent(String contentId, boolean reload)	Retourne le contenu ayant pour identificateur contentId. Le deuxième paramètre indique s'il faut recharger tous les contenus du dépositaire dans la mémoire ou non.
Content getContent (String key, String value, boolean reload)	Retourne le contenu ayant l'élément formé par le couple {*key*, *value*} comme sous-élément du descripteur
Content[] getContent (boolean reload)	Retourne tous les contenus du dépositaire, le paramètre *reload* permet de spécifier si le dépositaire doit être rechargé en mémoire ou non.
boolean removeContent (String contented)	Permet de supprimer le contenu ayant pour identificateur *contentId*, s'il existe.

Tableau 4.8. Méthodes offertes par l'interface *ContentManager*

Module demandeur proactif

En se basant sur les services de communication fournis par la couche Pair-à-Pair, ce module offre une API permettant à un nœud d'avoir un comportement *demandeur proactif*. Un tel demandeur se connecte à un fournisseur et échange avec lui des messages en vue de rapatrier les objets demandés. L'objet est soit l'instance de *Content*, soit l'entité représentée par le contenu (par exemple un fichier d'archive).

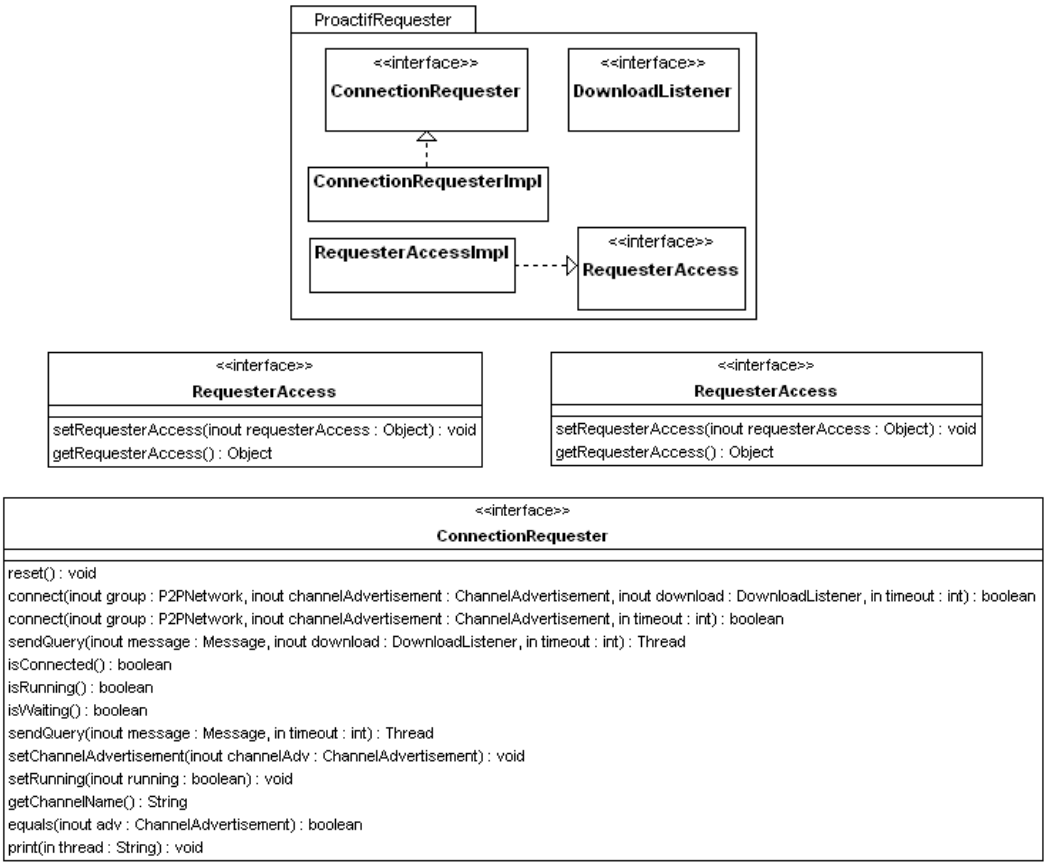


Figure 4.7. Module de communication pour un demandeur proactif

Avec la version 2 de JXTA, les échanges de messages entre les pairs se font à travers des sockets. Du coté du fournisseur, nous retrouvons *JXTAServerSocket*. Du coté du demandeur, nous utilisons la classe *JXTASocket*. Les attributs de ces classes se composent essentiellement du groupe de pairs auquel appartiennent les nœuds et du *PipeAdvertisement*, l'annonce relative au canal de communication. Afin d'abstraire ces entités au niveau de la couche noyau générique, nous introduisons *ProviderAccess* pour *JXTAServerSocket*, *RequesterAccess* pour *JXTASocket* et *ChannelAdvertisement* pour *PipeAdvertisement*. Dans ce qui suit, nous allons détailler l'API relative au demandeur proactif :

- ***RequesterAccess.*** Cette interface offre deux opérations :

Méthodes	**Description**
void setRequesterAccess (Object requesterAccess)	Permet d'initialiser l'attribut *JXTASocket* de la classe *RequesterAccessImpl.*
Object getRequesterAccess ()	Retourne une instance de *JXTASocket.*

Tableau 4.9. Méthodes offertes par l'interface *RequesterAccess*

- ***ConnectionRequester.*** Cette interface contient les opérations nécessaires du côté du demandeur pour communiquer avec le fournisseur :

Méthodes	**Description**
RequesterAccess connect (P2PNetwork group, ChannelAdvertisement channelAdvertisement, int timeout)	Permet d'établir une communication directe entre le demandeur et le fournisseur. Si le fournisseur ne donne pas de signe avant la fin du *timeout*, la connexion est annulée.
RequesterAccess connect (P2PNetwork group, ChannelAdvertisement channelAdvertisement, DownloadListener download, int timeout)	Permet d'établir une communication directe avec le fournisseur en spécifiant le traitement de l'objet demandé.
Thread sendQuery (Message message, int timeout)	Envoie un message au fournisseur. Retourne un *Thread* qui attend la réponse du fournisseur. L'attente est bornée (*timeout).*
Thread sendQuery (Message message, DownloadListener download, int timeout)	Envoie un message au fournisseur en spécifiant le traitement de l'objet demandé.
Boolean equals (ChannelAdvertisement adv)	Retourne *true* si l'annonce du canal de communication est égale à la valeur passée en parameter.
void setChannelAdvertisement(ChannelAdvertisement channelAdv)	Initiale l'annonce du canal de communication.
Boolean isConnected ()	Reourne *true* s'il y a une connexion avec le fournisseur.

Tableau 4.10. Méthodes offertes par l'interface *ConnectionRequester*

- ***DownloadListener.*** Cette interface permet de personnaliser les traitements lors de la réception du message provenant du fournisseur.

Méthodes	**Description**
void processMessage (Message message)	Cette fonction est appelée par le T*hread* qui est en attente de la réponse du fournisseur. Le message reçu est passé en paramètre.
void fail ()	Cette opération est appelée lorsqu'il y échec de réception.

Tableau 4.11. Méthodes offertes par l'interface *DownloadListener*

Le code suivant illustre comment personnaliser les traitements du message.

Même traitement

```
public class ProgrammeDemandeur {
   public static void main (String args[]) {
      //Message contenant l'identificateur de l'objet à rapatrier
      Message query;
      …
      //Créer une session de communication directe avec un fournisseur
      ConnectionRequester connectionReq = new ConnectionRequesterImpl();
      connectionReq.connect(group,
                            channelAdv,
                            new TheDownloadListener(),
                            connectionTimeout);
      if(connectionReq.isConnected()) {
         //Envoyer le message au fournisseur et attendre la réponse
         //Si pas de réponse après 20s, le demandeur fait autre chose
         connectionReq.sendQuery(query1, 20000);
         ...
         connectionReq.sendQuery(query2, 20000);
      }
```

```
        }

        class TheDownloadListener implements DownloadListener {
           public void processMessage(Message response) {
              //Traitement personnalisé de l'objet reçu
              …
           }
        }
     }
Différent traitements

     public class ProgrammeDemandeur {
        public static void main (String args[]) {
           //Message contenant l'identificateur de l'objet à rapatrier
           Message query;
           …
           //Créer une session de communication directe avec un fournisseur
           ConnectionRequester connectionReq = new ConnectionRequesterImpl();
           connectionReq.connect(group, channelAdv, connectionTimeout);
           if(connectionReq.isConnected()) {
              //Envoyer le message au fournisseur et attendre la réponse
              connectionReq.sendQuery(query, new TheDownloadListener1(), 20000);
              ...
              connectionReq.sendQuery(query, new TheDownloadListener2(), 20000);
           }
        }

        class TheDownloadListener1 implements DownloadListener {
           public void processMessage(Message response) {
              //Traitement personnalisé de l'objet reçu
              //Les instructions doivent être écrites dans cette partie

              …
           }
        }

        class TheDownloadListener2 implements DownloadListener {
           …
        }
     }
```

Figure 4.8. Personnalisation du traitement du côté Demandeur

Fonctionnement du demandeur

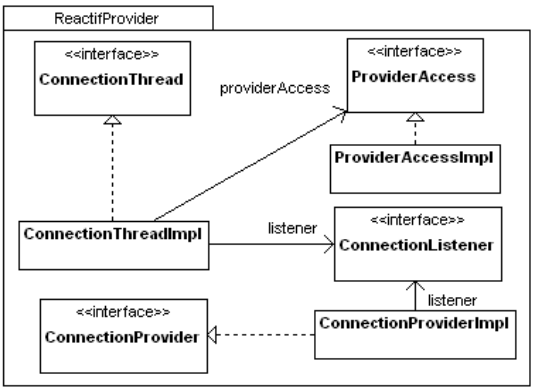
Après avoir instancié *ConnectionRequesterImpl*, le demandeur se connecte au fournisseur avec la méthode *connect()*. Une fois que la connexion est établie, le demandeur envoie la requête de rapatriement avec la méthode *sendQuery()* en spécifiant le délai maximum pour récupérer l'objet demandé. Le traitement de cet objet se fait en appelant la méthode *processMessage()* de la classe *TheDownloadListener*. Il peut être le même pour tous les objets ou bien personnalisé par objet. Pour cette raison, deux modes de traitement sont offerts :

- **Même traitement.** Dans ce cas, il suffit d'avoir une seule instance de *TheDownloadListener*.

- **Différent traitements**. Dans ce cas, il est nécessaire d'avoir une instance de *TheDownloadListener* pour chaque requête de rapatriement.

Module fournisseur réactif

Un nœud peut avoir un comportement *fournisseur réactif* en utilisant l'API fournie par ce module. Il répond à une requête de rapatriement en envoyant l'objet demandé.



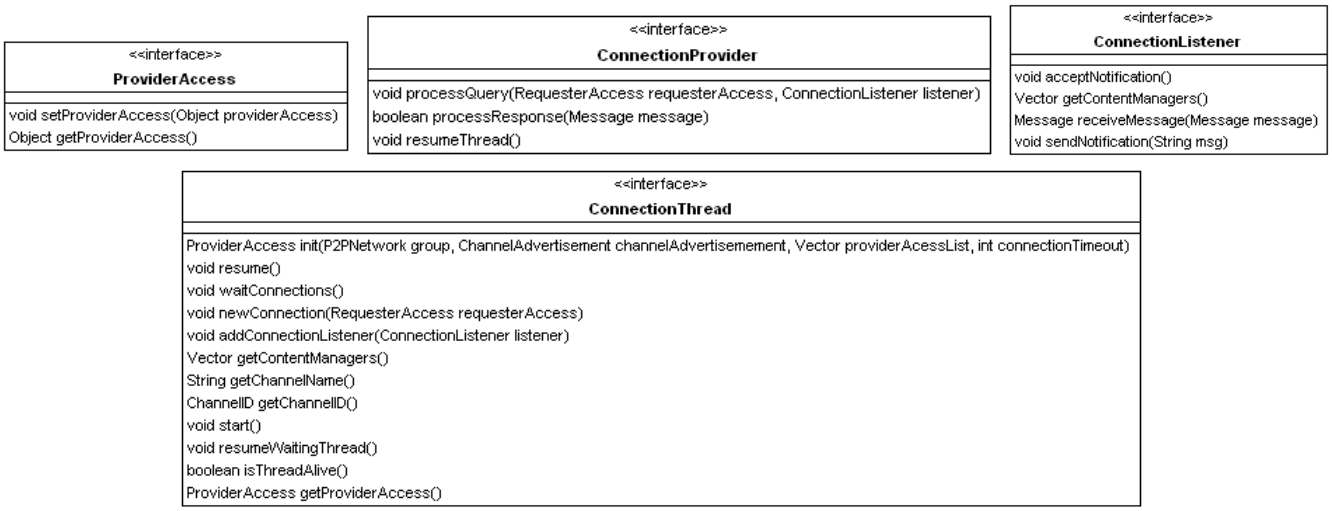


Figure 4.9. Module de communication pour un fournisseur réactif

- ***ProviderAccess.*** Cette interface permet d'abstraire *JXTAServerSocket*. Les opérations offertes sont :

Méthodes	**Description**
void setProviderAccess (Object requesterAccess)	Permet d'initialiser l'attribut *JXTAServerSocket* de la classe *ProviderAccessImpl.*
Object getProviderAccess ()	Retourne une instance de *JXTAServerSocket.*

Tableau 4.12. Méthodes offertes par l'interface *ProviderAccess*

- ***ConnectionThread*** offre le moyen de mettre en place un processus qui attend les requêtes provenant des demandeurs. Ce processus peut être lié à un ou plusieurs dépositaires où il pourra rechercher le contenu approprié lorsqu'il s'agit d'une demande de contenu.

Méthodes	**Description**
ProviderAccess init (P2PNetwork group, ChannelAdvertisement channelAdvertisemement, Vector providerAcessList, int connectionTimeout)	Permet d'initialiser le processus qui va traiter les demandes de rapatriements. Au delà de la durée définie par *connectionTimeout*, s'il n'y a pas une requête du coté du demandeur, la connexion déjà ouverte sera annulée.
Void waitConnections ()	Démarre le processus. Cette fonction fait appel à newConnection() pour chaque demande de connexion du côté du demandeur.
Void newConnection (RequesterAccess requesterAccess)	Permet de traiter les requêtes du demandeur dans une session de communication séparée.
Void addConnectionListener (ConnectionListener listener)	Ajoute la classe *listener* responsable de la personnalisation des traitements lors de la réception d'une demande de rapatriement.
Void resumeWaitingThread ()	Permet d'arrêter le processus principal ainsi que les autres processus qui communiquent avec les demandeurs.
boolean isThreadAlive ()	Retourne l'état du processus.

Tableau 4.13. Méthodes offertes par l'interface *ConnectionThread*

- ***ConnectionProvider***. Cette interface sert à communiquer avec le demandeur dans une session de communication séparée. Ceci permet d'ouvrir plusieurs communications parallèles et donc d'autoriser le traitement de plusieurs requêtes à la fois.

Méthodes	**Description**
Void processQuery (RequesterAccess requesterAccess, ConnectionListener listener, int bufferSize)	Permet de lancer un *Thread* qui fait appel à *receiveMessage(messagein)* de ConnectionListener pour traiter la requête du demandeur. Le message retourné est passé en paramètre à la fonction *processResponse(messageout).*
boolean processResponse (Message message)	Envoie le message vers le demandeur.

Tableau 4.14. Méthodes offertes par l'interface *ConnectionProvider*

- ***ConnectionListener*** doit être implantée au niveau de la couche plus haute et passée en paramètre à la fonction *processQuery*.

Méthodes	**Description**
Message receiveMessage (Message message)	Reçoit le message contenant la requête du demandeur et retourne le message contenant l'élément à envoyer au nœud demandeur
Void acceptNotification ()	Cette fonction est appelée par *ConnectionProviderImpl* lors de l'envoi de la réponse au demandeur
Void acceptNotification ()	Cette fonction est appelée par *ConnectionThreadImpl* lorsqu'il y a une ouverture de communication avec le demandeur.
Vector getContentManagers ()	Retourne un tableau des gestionnaires de dépositaires qui sont sollicités par le processus *ConnectionThread*.

Tableau 4.15. Méthodes offertes par l'interface *ConnectionListener*

Le code suivant explique l'implantation d'un programme fournisseur de contenus.

```
public class ProgrammeFournisseur {
   public static void main (String args[]) {
      //Mise en place d'un serveur en attente
      ConnectionThread connectionThread = new ConnectionThreadImpl();

      //Dans le cas d'une session de communication avec un demandeur,
      //si le fournisseur ne reçoit aucune requête pendant 30s,
      //la communication est ignorée
      connectionThread.init(group, channelAdv, 30000);
      //repositories est un vecteur de ContentManager
      connectionThread.addConnectionListener(new TheConnectionListener(repositories));
      connectionThread.start();
   }

   class TheConnectionListener implements ConnectionListener {
      public Message receiveMessage(Message query) {
         Message response;
         //Traitement personnalisé de la reqûete
         …
         //Retourne le message contenant l'objet demandé
         return response;
      }
   }
}
```

Figure 4.10. Personnalisation du traitement du côté Fournisseur

Fonctionnement du fournisseur

Pour démarrer le fournisseur, il suffit d'instancier *ConnectionThreadImpl*, de l'initialiser avec les paramètres adéquats et d'appeler la méthodes *start()*. Quand une requête lui parvient, la méthode *receiveMessage()* est appelée. Cette dernière retourne l'objet demandé s'il existe, enveloppé dans un autre objet de type *Message*.

4.3.3 Couche déploiement

La troisième couche de notre architecture se focalise sur le déploiement sensible au contexte. Elle utilise les modules offerts par la couche noyau générique pour exporter une API spécialisée dans le déploiement. Cette API est utilisée par la couche application pour développer des applicatifs faisant appel aux services de déploiement.

Dans la première section de ce chapitre, nous avons identifié les fonctionnalités que doit offrir une plate-forme de déploiement supportant l'algorithme proposé. La couche déploiement permet ainsi de réaliser ces fonctionnalités à travers des modules offrant le moyen de :

1. Se connecter (et se déconnecter) au groupe de déploiement,
2. Décrire, publier, découvrir et rapatrier des contenus modélisant les applications collaboratives, les descripteurs de contexte et les descripteurs des applications déployées,
3. Réaliser les opérations de déploiement et de reconfiguration.

Dans la suite, nous allons détailler ces modules :

Module gestion du réseau Pair-à-Pair

Le module *Gestion du réseau Pair-à-Pair* propose une API pour la gestion du réseau Pair-à-Pair. Il simplifie encore plus le module *Accès au réseau Pair-à-Pair* de la couche noyau générique en cachant par exemple les opérations de création des annonces des groupes de pairs et de découverte du groupe en ligne. La principale interface *P2PManager* permet en particulier d'initialiser l'application P2P, de se connecter (et de se déconnecter) au groupe de déploiement.

Module gestion des applications

Ce module offre une API qui se focalise sur la création des contenus représentant des applications. Il permet d'assister les développeurs dans la description des applications selon le modèle introduit dans la section 3.3.2 du troisième chapitre. Pour cela, il utilise principalement le module *Accès au dépositaire* de la couche noyau générique. L'interface *RequiredResource* abstrait n'importe quelle ressource requise par une application. Une ressource admet plusieurs attributs auxquels on peut accéder et que l'on peut modifier. Les opérations nécessaires pour la création des contenus d'application sont fournies par l'interface *ApplicationManager*.

Module gestion du contexte

Ce module offre une API qui se focalise sur la création des contenus représentant l'environnement d'exécution. Il permet d'assister les développeurs dans la description de l'état d'un nœud de déploiement selon le modèle introduit dans la section 3.3.3. Pour cela, il utilise principalement le module *Accès au dépositaire* de la couche noyau générique. De la même façon que pour le module *Gestion des applications*, l'interface *OfferedResource* abstrait n'importe quelle ressource offerte par l'environnement d'exécution. Conformément au modèle du nœud de déploiement, une ressource admet deux attributs le nom et la valeur, auxquels on peut accéder et que l'on peut modifier. Les opérations nécessaires pour la création des contenus du contexte d'exécution sont fournies par l'interface *ContextManager*.

Module gestion des implantations déployées

A la fin de l'exécution du processus du déploiement et de reconfiguration, un nœud de déploiement se retrouve avec un certain nombre d'implantations conformément au modèle de session en cours. Pour que les nœuds voisins puissent s'informer de son état et ainsi vérifier l'interopérabilité, le nœud de déploiement publie un contenu décrivant les configurations de déploiement. Le module gestion des implantations déployées permet de créer ce contenu et de l'enregistrer dans un dépositaire local. Pour cela, il utilise principalement le module *Accès au dépositaire* de la couche noyau générique. La classe *Implementation* est utilisée pour abstraire une implantation d'application réelle permettant ainsi d'accéder à ses attributs. L'interface *DeployedImplementationManager* ainsi que son implantation fournissent les opérations nécessaires pour créer les contenus décrivant l'état du nœud de déploiement par rapport aux implantations déployées.

Module gestion distribuée des applications

Ce module est composé d'une seule interface et de son implantation. Il offre une API pour gérer les opérations distribuées relatives aux applications comme la publication et la

découverte. Il permet en outre de mettre en place un processus pour servir les demandes de rapatriements des applications. Les modules utilisés par cette API sont : le module *Fournisseur réactif*, le module *Accès au réseau Pair-à-Pair*, et le module *Gestion des applications*.

- ***ApplicationProvider.*** Cette interface et son implantation permettent la gestion distribuée des applications à travers les fonctions citées dans le tableau suivant :

Méthodes	**Description**
ConnectionThread waitRequests (P2PNetwork group, String channel, Vector applicationMngs, int timeout, int connectionTimeout)	Permet de mettre en place un processus en écoute des requêtes de rapatriement. Le paramètre *channel* spécifie le nom du canal qui va être utilisé pour recevoir les messages des demandeurs. Le tableau *applicationMngs* fait référence aux dépositaires que le processus consulte pour récupérer les contenus recherchés. *timeout* est utilisé pour fixer la durée limite pour découvrir l'annonce du canal. *connectionTimeout* spécifie la durée au delà de laquelle s'il n'y a pas une requête du côté du demandeur, la connexion déjà ouverte sera annulée.
Boolean publishApplications (P2PNetwork group, ConnectionThread connectionThread)	Publie tous les contenus disponibles dans les dépositaires attachés à *connectionThread*.
Boolean publishApplication (P2PNetwork group, ConnectionThread connectionThread, Content application)	Publie le contenu d'application en l'attribuant à *connectionThread*.
ContentAdvertisement createApplicationAdvertisement (P2PNetwork group, Content content, ChannelID channelId, String channelname)	Retourne une annonce d'un contenu passé en paramètre. Le fournisseur publiant le contenu est reconnaissable par le nom et l'identificateur du canal de communication.
void listOnlineThreads ()	Permet d'énumérer les processus qui sont en ligne
void removeWaitingRequests (P2PNetwork group, ConnectionThread connectionThread)	Permet de supprimer le processus passé en paramètre
ContentAdvertisement[] searchApplication (P2PNetwork group, String datatype, String direction, int timeout, boolean flush)	Permet de découvrir les annonces des applications qui respectent les critères de recherche *datatype* et *direction*. La durée maximale de la découverte est bornée par *timeout*. Le paramètre *flush* indique s'il faut garder ou effacer les annonces précédemment découvertes.

Tableau 4.16. Méthodes offertes par l'interface *ApplicationProvider*

Module gestion distribuée des implantations déployées

Ce module est composé d'une seule interface et de son implantation. Il offre une API pour gérer les opérations distribuées relatives aux implantations déployées comme la publication et la découverte. Il permet en outre de mettre en place un processus pour servir les demandes de rapatriements des contenus décrivant les implantations déployées. Les modules utilisés par cette API sont : le module *Fournisseur réactif*, le module *Accès au réseau Pair-à-Pair*, et le module *Gestion des implantations déployées*.

- ***DeployedImplementationProvider.*** Cette interface et son implantation permettent la gestion distribuée des implantations déployées à travers les fonctions citées dans le tableau suivant :

Méthodes	**Description**
ConnectionThread waitDeployedImplementationRequests (P2PNetwork group, String channel, Vector implementationMngs, int timeout, int connectionTimeout)	Permet de mettre en place un processus en écoute des requêtes de rapatriement. Le paramètre *channel* spécifie le nom du canal qui va être utilisé pour recevoir les messages des demandeurs. Le tableau *implementationMngs* fait référence aux dépositaires que le processus consulte pour récupérer les contenus recherchés. *timeout* est utilisé pour fixer la durée limite pour découvrir l'annonce du canal. *connectionTimeout* spécifie la durée au delà de laquelle s'il n'y a pas une requête du côté du demandeur, la connexion déjà ouverte sera annulée.
Boolean publishDeployedImplementationContent (P2PNetwork group, ConnectionThread connectionThread)	Publie tous les contenus disponibles dans les dépositaires attachés à *connectionThread*.
Boolean publishDeployedImplementationContent (P2PNetwork group,	Publie le contenu décrivant les implantations déployées, en l'attribuant à *connectionThread*.

ConnectionThread connectionThread, Content deployedImplementation)	
void publishDeployedImplementationContent (P2PNetwork group, ConnectionThread connectionThread, String sessionid)	Publie le contenu décrivant les implantation déployées relative à la session dont l'identificateur est passé en paramètre, en l'attribuant à *connectionThread*.
ContentAdvertisement createDeployedImplementationAdvertisement (P2PNetwork group, Content content, ChannelID channelId, String channelname)	Retourne une annonce d'un contenu passé en paramètre. Le fournisseur publiant le contenu est reconnaissable par le nom et l'identificateur du canal de communication.
Content convertImplementationToApplicationContent (Content content)	Permet de convertir un contenu décrivant une implantation vers un contenu décrivant une application
void listOnlineThreads ()	Permet d'énumérer les processus qui sont en ligne
void removeWaitingRequests (P2PNetwork group, ConnectionThread connectionThread)	Permet de supprimer le processus passé en paramètre
ContentAdvertisement searchDeployedImplementation (P2PNetwork group, String nodename, int timeout, boolean flush)	Permet de découvrir l'annonce du contenu décrivant les implantations déployées sur le nœud de déploiement dont le nom est passé en paramètre. La durée maximale de la découverte est bornée par *timeout*. Le paramètre *flush* indique s'il faut garder ou effacer les annonces précédemment découvertes.

Tableau 4.17. Méthodes offertes par l'interface *DeployedImplementationProvider*

Module contrôle du déploiement

Ce module implante l'algorithme de déploiement et de reconfiguration. Afin de permettre le déroulement de plusieurs tâches en même temps, certaines parties de l'algorithme - comme la découverte des applications et la collecte des documents décrivant les implantations sur les nœuds des voisins - sont exécutées dans des processus légers.

Notons que nous utilisons *JGraph* pour représenter graphiquement l'instance du modèle de session. Ce composant nous permet de parcourir rapidement le graphe de session afin d'extraire les informations utilisées dans le déploiement. Dans l'implantation actuelle, nous avons simulé les opérations de reconfiguration. Les résultats obtenus sont favorables, ce qui nous conforte nos choix dans l'implantation réelle de la reconfiguration. Dans la suite, nous allons décrire ce module.

- ***DeploymentEngine*** est le processus principal qui dirige les opérations de déploiement. Il accepte le graphe modélisant la session en cours, comme paramètre de son constructeur. Ensuite il fait appel aux autres classes selon l'ordre défini par l'algorithme.

Méthodes	**Description**
void identifyRequirements ()	Identifie les besoins du nœud de déploiement à partir du graphe de session. Pour chaque besoin, cette fonction crée une nouvelle instance de la classe *Requirement*.
void identifyNeighbours ()	Identifie les nœuds voisins du nœud de déploiement. Chaque besoin déterminé aura une liste de nœuds voisins.
void deployImplementation ()	Pour chaque besoin déterminé, cette méthode démarre un nouveau *thread* responsable de retrouver les implantations qui couvrent le besoin.
void retrieveDeployedImplementationContent ()	Permet de lancer un *thread* responsable de récupérer les contenus décrivant les implantations déployées sur les nœuds voisins.

Tableau 4.18. Méthodes offertes par la classe *DeploymentEngine*

- ***ImplementationDeployer*** étend la classe *Thread.* Ainsi, pour chaque besoin identifié, un *thread* est démarré dans le but est de rechercher les configurations de déploiement valides. Dans la méthode *run()*, le processus commence d'abord par rechercher les implantations candidates parmi celles qui sont déjà déployées sur le nœud de déploiement. Si le résultat n'est pas null, alors il poursuit le reste de l'algorithme de déploiement. Sinon, il démarre un autre *thread* responsable de faire la découverte des

applications dans le groupe de pairs et il poursuit les opérations de déploiement, en particulier le rapatriement des archives des implantations sélectionnées. Pour cela, la classe *TheDownloadListener* implante l'interface *DownloadListener* et définit le traitement personnalisé lors de la réception de l'archive d'une implantation. Dans le tableau suivant, nous allons énumérer les principales méthodes de la classe *ImplementationDeployer*.

Méthodes	**Description**
Boolean localDiscover ()	Permet de découvrir les implantations candidates parmi celles déjà déployées sur le nœud de déploiement.
void verifyLocalCompatibility ()	Pour chaque application découverte, cette méthode permet de ne retenir que les implantations compatibles avec le contexte d'exécution.
boolean match (ContentDescriptor implementation, ContentDescriptor context)	Permet de confronter le descripteur d'implantation avec le descripteur du contexte d'exécution. La méthode retourne *true* si les ressources offertes couvrent les ressources exigées par l'implantation.
void identifyInteroperableNeighbour()	Permet d'attribuer à chaque implantation compatible, une liste des nœuds offrants des implantations interopérables.
int generateConfigurations ()	Génère toutes les configurations de déploiement possibles avec les implantations qui sont compatibles et interopérables.
int downloadConfiguration ()	Permet de choisir une configuration valide et de rapatrier les archives des implantations appropriées.
void instantiateImplementations ()	Permet d'instancier les implantations rapatriées.

Tableau 4.19. Méthodes offertes par la classe *ImplementationDeployer*

- ***TheDownloadListener* imbriqué dans la classe *ImplementationDeployer.*** Cette classe implante l'interface *DownloadListener*. Elle définit le traitement à faire lors de la réception d'un message provenant du fournisseur d'application. Ce message contient soit un message d'erreur, soit l'archive de l'implantation appropriée. Dans ce cas, le fichier jar sera enregistré dans un répertoire local.

Méthodes	**Description**
Void processMessage(Message message)	Permet de traiter le message du fournisseur passé en paramètre. Si le message contient l'archive d'une implantation, cette dernière sera enregistrée dans un répertoire.

Tableau 4.20. Méthodes offertes par la classe *TheDownloadListener*

- ***ApplicationDiscoverer.*** Cette classe étend aussi la classe *Thread.* Elle se préoccupe de découvrir les applications couvrant le besoin passé en paramètre dans son constructeur. Connaissant le type de données et le sens de communication, le *thread* commence d'abord par rechercher les annonces des applications. Pour cela, il utilise la méthode *searchApplication()* du module *ApplicationDistributedManagement.* Chaque annonce découverte est parcourue afin d'en extraire l'identificateur et le nom du canal de communication. Ces deux variables seront utilisées pour se connecter au fournisseur de l'application afin de rapatrier le l'objet *Content* approprié s'il n'a pas été déjà rapatrié précédemment. Le message envoyé au fournisseur avec la méthode *sendQuery()* de *ConntectionRequester*, contient l'identificateur de l'application qui est aussi récupéré à partir de l'annonce. Cette méthode retourne un *thread*. Ainsi, chaque rapatriement est effectué dans un processus séparé. Le traitement personnalisé du contenu rapatrié est codé dans la méthode *processMessage()* de la classe *TheDownloadListener* qui implante l'interface *DownloadListener*. Cette classe imbriquée dans la classe *ApplicationDiscoverer,* est passé en paramètre de la méthode

sendQuery() et sera invoquée lors de la réception du message provenant du fournisseur.

Méthodes	Description
void run ()	Permet de découvrir les applications candidates parmi celles publiées dans le réseau Pair-à-Pair.
Vector getDiscovered ()	Retourne un tableau des contenus rapatriés.

4.21. Méthodes offertes par la classe *ApplicationDiscoverer*

- ***TheDownloadListener* imbriqué dans la classe *ApplicationDiscoverer.*** Cette classe implante l'interface *DownloadListener*. Elle définit le traitement à faire lorsque le demandeur d'application reçoit un message du fournisseur d'application. Le message contient soit un message d'erreur, soit le contenu approprié. Dans ce cas, le contenu d'application sera enregistré dans un dépositaire local. Ensuite, il sera parcouru pour extraire les implantations qui vérifient le contexte d'exécution propre au nœud de déploiement.

Méthodes	Description
void processMessage(Message message)	Permet de traiter le message du fournisseur passé en paramètre.
void verifyLocalCompatibility (Content application)	Permet de vérifier la compatibilité des implantations appartenant à l'application dont le contenu est passé en paramètre
Enumeration getCompatibleImplementation (Content application)	Retourne les implantations compatibles de l'application dont le contenu est passé en paramètre
boolean match (ContentDescriptor implementation, ContentDescriptor context)	Permet de confronter le descripteur d'implantation avec le descripteur du contexte d'exécution. La méthode retourne *true* si les ressources offertes couvrent les ressources exigées par l'implantation.
boolean ressourceTest (String Type, String requiredvalue, String offeredValue, String Operator)	Permet de comparer la valeur exigée d'une ressource avec celle offerte.
ContentDescriptor createImplementationDescriptor (String applicationId, String category, Element name, Element dataExchanged, Element implementation)	Permet de décrire une implantation en lui ajoutant les informations concernant les données échangées.

Tableau 4.22. Méthodes offertes par la classe *TheDownloadListener*

- ***DeployedImplementationContentCollector*.** Cette classe parcourt les besoins du nœud de déploiement dans le but d'identifier les nœuds voisins. Pour chaque nœud voisin, elle démarre un nouveau *thread* qui se charge de rapatrier le contenu décrivant les implantations déployées sur le nœud voisin.

Méthodes	Description
void run ()	Permet de parcourir les besoins du nœud de déploiement pour déterminer les nœuds voisins et procéder au rapatriement du contenu décrivant les implantations déployées sur chaque voisin.

Tableau 4.23. Méthodes offertes par la classe *DeployedImplementationContentCollector*

- ***DeployedImplementationContentDiscoverer***. Cette classe étend la classe *Thread*. L'appel à la méthode *run()*, permet d'exécuter les opérations nécessaires pour rapatrier le contenu décrivant les implantations déployées sur le nœud voisin dont l'identité est passée en paramètre dans le constructeur. Ainsi, le *thread* commence d'abord par découvrir l'annonce du contenu en utilisant la méthode *searchDeployedImplementation()* du module *DeployedImplementationDistributedManagement*. Ensuite, à partir de l'annonce

retrouvée, il extrait le nom et l'identificateur du canal pour se connecter au nœud voisin. Avec la méthode *sendQuery()* de *ConnectionRequester*, il envoie une requête de rapatriement. Le traitement personnalisé du message reçu, est défini dans le code de la méthode *processMessage()* de la classe *TheDownloadListener* implantant l'interface *DownloadListener*.

Méthodes	**Description**
void run ()	Procède à la découverte de l'annonce du contenu décrivant les implantations puis au rapatriement du contenu.

Tableau 4.24. Méthodes offertes par la classe *DeployedImplementationContentDiscoverer*

- ***TheDownloadListener* lié à la classe *DeployedImplementationContentDiscoverer.*** Une fois que le message est parvenu au nœud de déploiement, la méthode *processMessage()* permet de le parcourir afin de récupérer le contenu décrivant les implantations déployées sur le nœud voisin. Ce contenu sera utilisé par la suite pour vérifier la contrainte d'interopérabilité.

Méthodes	**Description**
Void processMessage(Message message)	Permet de traiter le message du fournisseur passé en paramètre.

Tableau 4.25. Méthodes offertes par la classe *TheDownloadListener*

- ***DeploymentEngineSimulation.*** Cette classe regroupe les principales fonctionnalités offertes par les classes précédentes dans un même processus principal permettant de simuler le déploiement. Le processus accepte le graphe modélisant la session en cours, comme paramètre de son constructeur. Ensuite, il effectue les étapes de l'algorithme de déploiement. Plus de détails de la simulation sont donnés dans la section 5.2.

Méthodes	**Description**
void identifyRequirements ()	Identifie les besoins du nœud de déploiement à partir du graphe de session. Pour chaque besoin, cette fonction crée une nouvelle instance de la classe *Requirement.*
void identifyNeighbours ()	Identifie les nœuds voisins du nœud de déploiement. Chaque besoin déterminé aura une liste de nœuds voisins.
boolean localDiscover (JGraph currentGraph, Terminal terminal)	Permet de découvrir les contenus des applications respectant les critères de recherche pour chaque besoin identifié.
void collectContext (JGraph currentGraph, Terminal terminal)	Permet de découvrir les contenus décrivant les implantations déployées sur chaque nœud voisin.
void verifyLocalCompatibility (JGraph currentGraph, Terminal terminal)	Pour chaque application découverte, cette méthode permet de ne retenir que les implantations compatibles avec le contexte d'exécution.
void identifyInteroperableNeighbour (JGraph currentGraph, Terminal terminal)	Permet d'attribuer à chaque implantations compatible, une liste des nœuds offrants des implantations interopérables.
int generateConfigurations(JGraph currentGraph, Terminal terminal)	Génère toutes les configurations de déploiement possibles avec les implantations qui sont compatibles et interopérables.
int downloadConfiguration(JGraph currentGraph, Terminal terminal)	Simule le rapatriement des implantations. Il s'agit de copier les contenus des implantations de la configuration choisi, dans un répertoire spécifique.
int instantiateApplications(JGraph currentGraph, Terminal terminal)	Simule l'instantiation des implantations. Il s'agit d'attribuer le nombre d'instantiation nécessaire au contenu de chaque implantation.

Tableau 4.26. Méthodes offertes par la classe *DeploymentEngineSimulation*

- ***Planner.*** Le but principal de cette classe est de générer des configurations de déploiement valides. Pour chaque besoin identifié et à partir des implantations candidates, le *planner* construit toutes les combinaisons possibles pour former les configurations de déploiement tout en attribuant un poids comme décrit dans la partie dédiée à l'algorithme de déploiement. Ensuite, il parcourt la liste générée en vérifiant la validité de chaque configuration. Les principales méthodes de cette classe sont :

Méthodes	Description
void addImplementation (Implementation implementation)	Rajoute une implantation à la liste des implantations candidates
Implementation getImplementation (int index)	Retourne une implantation
void generate ()	Permet de générer toutes les configurations possibles
void validateConfigurations ()	Permet de valider les configurations générées
Configuration firstValidConfiguration ()	Retourne la configuration optimal

Tableau 4.27. Méthodes offertes par la classe *Planner*

- ***Requirement***. Cette classe abstrait un besoin du nœud de déploiement. Les principaux attributs sont :

Attributs	Description
String id ;	Identifie le besoin et le distingue des autres besoins
String dataType ;	Type de données : {Video, Audio, Text, …}
String direction ;	Sens de communication : {producer, consumer}
Vector neighbours ;	Tableau des nœuds voisins restreint à ce besoin.
Vector discoveredApplications ;	Tableau des contenus des applications découvertes
Vector deployedImlementations ;	Tableau des contenus des implantations déployées
Vector implementations = new Vector()	Tableau des contenus des implantations candidates
Vector dataflows ;	Tableau des noms des flux de données restreint au besoin
Planner configurations ;	Générateur des configurations de déploiement.
boolean deployed ;	Indique si le besoin a été traité avec succès ou pas

Tableau 4.28. Attributs offerts par la classe *Requirement*

Les méthodes de cette classe se focalisent particulièrement sur l'initialisation, l'ajout et la récupération des valeurs de ces attributs.

- ***Configuration*** sert pour décrire une configuration de déploiement. Les attributs de cette classe sont :

Attributs	Description
int[] configuration ;	Donne les index des implantations formant la configuration
boolean valid ;	Indique si la configuration est valide ou non
int weight ;	Donne le poids de la configuration

Tableau 4.29. Attributs offerts par la classe *Configuration*

Les méthodes de cette classe servent à initialiser ces attributs et à les récupérer.

- ***Dataflow*** représente un flux de données et les nœuds voisins échangeant ce flux (voir colonne 3 du tableau 3.1 du troisième chapitre) :

Attributs	Description
String flowname ;	Définit le nom du flux de données
Vector neighbours ;	Tableau des nœuds voisin échangeant le flux de données

Tableau 4.30. Attributs offerts par la classe *Dataflow*

De la même façon, les méthodes servent à l'initialisation, à l'ajout et à la récupération des valeurs des attributs de cette classe.

- ***PreDynamicDeploymentSimulation***. Cette classe simule la première étape de la reconfiguration. Elle permet donc de déterminer le sous-ensemble pour la reconfiguration et l'organisation de ce sous-ensemble. Elle accepte le graphe modélisant la nouvelle session et le parcourt en recherchant les nœuds qui nécessitent des mises à jour. Nous utilisons cette classe dans le simulateur que nous avons développé au niveau de la couche application. Pour chaque nœud de la session, une instance de cette classe qui étend la classe *Thread*, est démarrée. Après la réorganisation du sous-ensemble des nœuds qui nécessitent une reconfiguration, si le

thread correspond au premier nœud du sous-ensemble, alors il fait appel à la classe *DynamicDeploymentSimulation*, sinon il sera détruit :

Méthodes	**Description**
void identifyRequirements (JGraph designGraph, EllipseCell node)	Identifie les besoins d'un nœud du graphe modélisant la nouvelle session
void identifyNeighbours (JGraph designGraph, EllipseCell node)	Détermine les nœuds voisins du nœud dont l'identité est passé en paramètre (*node*)
boolean requireReConfiguration (EllipseCell simulatedNode, EllipseCell newSimulatedNode, boolean previousdeployment)	En comparant l'état précédent de la session et son nouvel état, cette méthode retourne *true* si le nœud requiert un reconfiguration, *false* sinon.
Vector searchSubGroup (JGraph designGraph, JGraph simulatedGraph, EllipseCell newSimulatedNode)	Retourne un tableau contenant le sous-ensemble des nœuds qui nécessitent une reconfiguration.
Vector orderSubGroup (Vector subgroup)	Réorganise le sous-ensemble des nœuds qui nécessitent une reconfiguration.

Tableau 4.31. Méthodes offertes par la classe *PreDynamicDeploymentSimulation*

- ***DynamicDeploymentSimulation***. Cette classe étend aussi la classe *Thread*. Elle permet de simuler la deuxième étape de la reconfiguration. Seul le nœud ayant le jeton, démarre le *thread*. Ainsi pour chaque besoin, la simulation du déroulement des opérations de déploiement se fait selon les six cas possibles énumérés dans la spécification de l'algorithme (voir section 3.4.3) La recherche des applications appropriées se fait en parcourant les contenus enregistrés dans un dépositaire local et non en envoyant une requête dans le réseau. Le fonctionnement est identique pour le déploiement : il s'agit de mettre à jour les attributs de *Requirement* sans lancer un rapatriement réel. Nous nous limitons aux principales méthodes qu'offre cette classe.

Méthodes	**Description**
Enumeration searchApplication (String datatype, String direction)	Retourne un tableau des contenus d'applications qui répondent aux critères de recherche {*datatype* et *direction*}. La recherche se fait en parcourant un dépositaire local.
void verifyLocalCompatibility (EllipseCell simulatedNode, Requirement requirement)	Détermine les implantations compatibles avec le contexte d'exécution.
boolean identifyInteroperableNeighbour (Vector neighbours, Vector reqsimulneib, Requirement requirement)	Pour chaque implantation, cette méthode identifie les voisins interopérables
int generateConfigurations(Requirement requirement)	Génère les configurations de déploiement en tenant en compte des implantations qui sont déjà déployées
int downloadConfiguration (EllipseCell simulatedNode, Requirement requirement)	Simule le rapatriement des implantations
int instantiateApplications(EllipseCell simulatedNode, Requirement requirement)	Simule l'instanciation des implantations en tenant en compte des implantations qui sont déjà déployées

Tableau 4.32. Méthodes offertes par la classe *DynamicDeploymentSimulation*

4.4 Conclusion

Nous avons présenté dans ce chapitre la conception de la plate-forme de déploiement permettant de supporter l'algorithme proposé. La première partie décrit donc les fonctionnalités que doit offrir une telle plate-forme. Nous avons identifié quatre fonctionnalités : accès au réseau P2P, interaction locale, interaction distribuée et contrôle de déploiement.

La deuxième partie a été consacrée à la description de la plate-forme développée dans le cadre de cette thèse. L'objectif principal de cette plate-forme est d'offrir des modules permettant l'implantation réelle de notre algorithme.

Le chapitre suivant montre l'étude de performance que nous avons menée pour évaluer d'une part l'algorithme de déploiement et d'autre part la plate-forme supportant cet algorithme.

Chapitre 5 Expérimentations

5.1 Introduction

Ce chapitre présente l'étude que nous avons menée pour l'évaluation de notre approche. Dans la littérature [46], l'évaluation peut se faire de deux manières : évaluation basée sur la perception de l'utilisation finale et évaluation basée sur les capacités du système développé. La première permet de mesurer la satisfaction de l'utilisateur par rapport aux fonctionnalités offertes par un système. La deuxième permet de comparer différents scénarios d'un système par rapport à un scénario de référence. Bien que l'évaluation selon la deuxième manière permette une confrontation plus objective des différentes mesures, la corrélation des résultats obtenus avec la satisfaction de l'utilisateur n'est pas toujours facilement interprétable.

Dans ce chapitre, nous avons mené nos évaluations selon les deux approches précédentes. Pour se rapprocher des évaluations utilisateurs, nous avons développé un ensemble de cinq prototypes qui caractérisent l'utilisation de nos concepts. Dans la partie suivante, nous procédons à des évaluations système dans laquelle nous nous focalisons sur l'évaluation basée sur les capacités de la plate-forme développée. Les expérimentations ont pour objectifs de :

- Evaluer les performances de l'algorithme de déploiement, particulièrement les étapes de vérification de la compatibilité avec le contexte d'exécution, de vérification de l'interopérabilité avec les nœuds voisins, de génération des configurations de déploiement, et de validation de ces configurations.

- Evaluer les modules de l'architecture permettant d'effectuer les échanges entre les nœuds. Le comportement fournisseur réactif ainsi que le comportement demandeur proactif seront analysés indépendamment du déploiement.

Pour chaque cas, nous allons détailler plus l'objectif de l'étude, décrire comment nous avons procédé, montrer les mesures obtenues sous forme de courbes, et enfin discuter ces résultats.

5.2 Utilisation de l'API de déploiement

Afin de s'assurer de la flexibilité d'utilisation de notre API, nous avons développé plusieurs logiciels applicatifs, présentés par la suite.

5.2.1 Simulateur de déploiement

A l'aide de l'API de déploiement, nous avons réalisé un prototype simulateur de déploiement dont le but est de tester l'algorithme proposé par simulation avant de passer aux

conditions réelles. Ce prototype permet de simuler d'une part le déploiement des applications suite à l'entrée d'un participant à la session, et d'autre part de procéder à la reconfiguration en réponse aux évolutions de la structure de la session. Nous avons donc utilisé les modules offerts par la couche déploiement de l'architecture proposée. Le prototype, quant à lui, se situe au niveau de la couche application. Les interactions entre nœuds sont simulées, puisque le programme opère localement. La plate-forme JXTA n'intervient donc pas dans la simulation. La figure 5.1 illustre une vue générale de ce prototype. La partie de gauche offre le moyen de créer un graphe décrivant la session selon le modèle introduit dans la section 3.3.1. La partie de droite permet de simuler les opérations de déploiement et de reconfiguration et d'avoir un aperçu graphique sur le résultat du déploiement. La partie inférieure affiche de façon textuelle, le déroulement des différentes opérations. Avec le simulateur, il est possible de créer des contenus représentant les applications et les contextes de déploiement selon les modèles des sections 3.3.2 et 3.3.3.

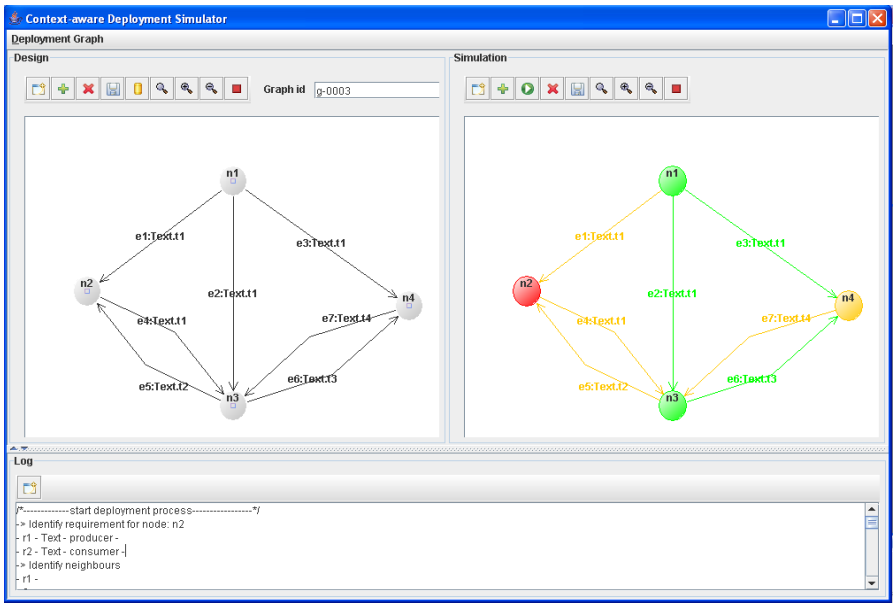


Figure 5.1. Simulateur de déploiement

Le simulateur de déploiement se compose essentiellement de cinq parties relatives à la gestion des applications, la gestion du contexte local, la conception de la structure de la session, la simulation du déploiement, et à la visualisation du résultat du déploiement et de la reconfiguration.

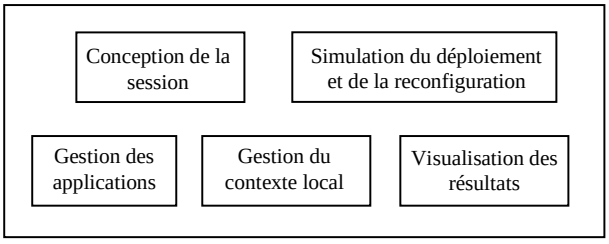


Figure 5.2. Composition du simulateur de déploiement

Gestion des applications

L'algorithme de déploiement se base sur la découverte des applications les plus aptes à couvrir les besoins des participants de la session. Dans le simulateur, cette fonctionnalité se réalise en effectuant une recherche dans un dépositaire local. Le prototype offre une fenêtre permettant de visualiser graphiquement les contenus des applications et d'effectuer des

opérations de gestion de contenus en se basant sur le module *Gestion des applications* de la couche déploiement. La figure 5.3 donne un aperçu de cette fenêtre.

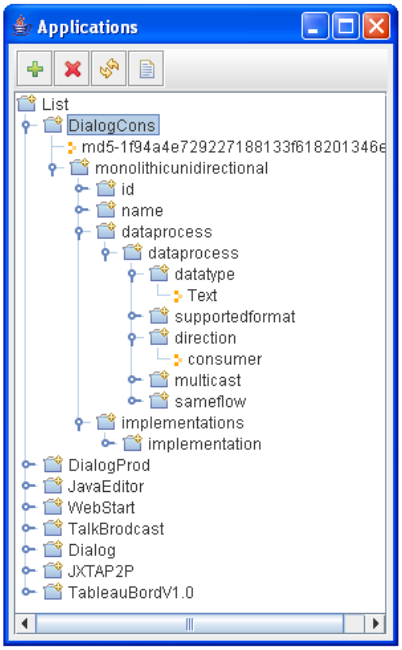


Figure 5.3. Fenêtre pour la gestion des descripteurs d'application

Les fonctions et les actions offertes par cette fenêtre sont :

- Affichage des contenus des applications sous forme d'arbre en se basant sur la bibliothèque JTree de Java. La racine de l'arbre pointe sur le dépositaire de contenus. Les nœuds intermédiaires sont les éléments composés d'un contenu et les feuilles représentent les éléments simples.
- Naviguer entre les éléments des descripteurs des applications et ainsi concevoir différents scénarios de déploiement en fonction des contenus disponibles.
- Sélection d'un contenu d'application pour l'afficher dans une autre fenêtre sous forme d'un fichier XML .
- Mise à jour des attributs des contenus des applications en saisissant la nouvelle valeur dans l'endroit approprié.
- Ajout d'un nouveau contenu conformément au modèle d'application. En cliquant sur le bouton d'ajout de contenu , une nouvelle fenêtre s'affiche proposant un formulaire à remplir. Le contenu généré sera enregistré dans le dépositaire local et affiché dans la fenêtre principale.
- Suppression d'un contenu . Cette action permet d'effacer le contenu du dépositaire et de le retirer de l'arbre.
- Rafraîchir l'arbre des contenus en cliquant sur le bouton approprié .

Gestion des contextes

Le choix des applications à déployer sur un nœud de déploiement dépend du contexte d'exécution. Le simulateur offre une fenêtre permettant de visualiser graphiquement les contenus décrivant des contextes d'exécution variés. Il se base sur le module *Gestion du contexte local* de la couche déploiement. La figure 5.4 donne un aperçu de cette fenêtre.

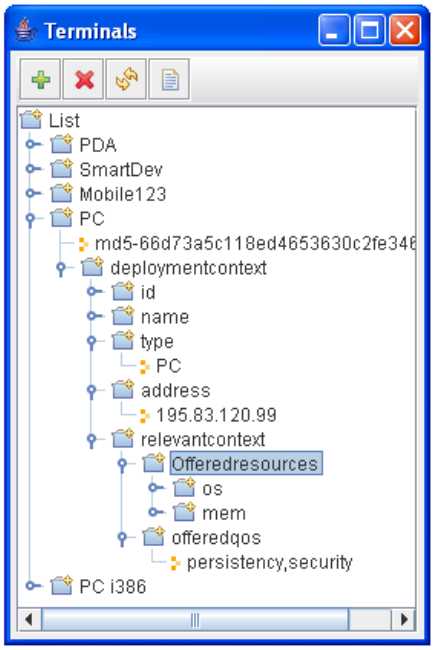


Figure 5.4. Fenêtre pour la gestion des descripteurs de contexte

De la même façon que pour la gestion des applications, les fonctions et les actions offertes par cette fenêtre sont :

- Affichage des contenus des contextes d'exécution sous forme d'arbre en se basant sur la bibliothèque JTree de Java.
- Navigation entre les éléments des descripteurs des contextes d'exécution permettant ainsi d'imaginer des scénarios de déploiement.
- Sélection d'un contenu pour l'afficher sous forme d'un fichier XML  .
- Mise à jour des attributs des contenus en saisissant la nouvelle valeur dans l'endroit approprié.
- Ajout d'un nouveau contenu conformément au modèle du nœud de déploiement. En cliquant sur le bouton d'ajout de contenu  , une nouvelle fenêtre s'affiche proposant un formulaire à remplir.
- Suppression d'un contenu avec le bouton  .
- Rafraîchir l'arbre des contenus avec le bouton  .

Conception de la session

L'objectif du prototype est de permettre la simulation du déploiement pour différents schémas de coopération. Pour cette raison, il offre une zone où un administrateur de session peut concevoir graphiquement l'instance du modèle de session sous forme d'un graphe orienté étiqueté. Par la suite, ce graphe peut être utilisé dans l'algorithme de déploiement (figure 5.1). Le prototype utilise principalement la bibliothèque JGraph pour afficher le graphe de session.

Ainsi, il est possible de faire les opérations suivantes :

- Ajout d'un nœud au graphe modélisant la session. Un label est attribué automatiquement pour chaque nœud ajouté.
- Suppression d'un nœud du graphe qui entraîne aussi la suppression de tous les arcs liés à ce nœud.
- Ajout d'un arc au graphe de session. Avec la souris, l'administrateur peut créer un nouvel arc entre deux nœuds du graphe. Pour cela, il positionne l'extrémité du pointeur de la souris sur le nœud source, puis il le fait glisser jusqu'au nœud cible, en maintenant le bouton gauche de la souris enfoncé. En relâchant le pointeur de la souris sur le nœud cible, une boîte de dialogue s'affiche pour choisir le type de données et saisir le nom du flux de données (figure 5.5).

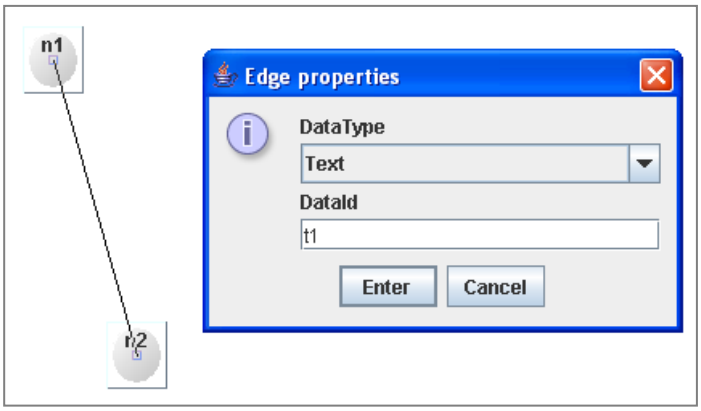


Figure 5.5. Ajout d'arc entre les nœuds du graphe de session

- Effacer un arc. Pour cela, il suffit de le sélectionner et d'utiliser la touche « supprime » du clavier ou le bouton Par ailleurs, le prototype offre la possibilité de sélectionner plusieurs nœuds et arcs pour les supprimer par la suite.
- Créer une nouvelle zone pour dessiner le graphe de session. Le bouton permet d'effacer l'ancien graphe et d'avoir une nouvelle zone vide.
- Charger un graphe depuis un fichier XML. Avec le bouton , il est possible de sélectionner un fichier du disque dur contenant l'instance du modèle de session sous forme d'un document XML. En parcourant ce fichier, le prototype ajoute les nœuds et les arcs décrivant la session dont on veut simuler le déploiement.

- Enregistrer le graphe conçu dans un document XML. Après avoir décidé du graphe de session, l'administrateur peut utiliser le bouton  , pour l'enregistrer sur le disque dur.

- Agrandir, diminuer la taille du graphe et revenir à la taille initiale avec les boutons  . Ceci offre l'avantage de visualiser des graphes de grande taille.

- Bloquer le graphe dessiné avec le bouton  . Cette action permet de réaliser le déploiement sans risque de changement au niveau du graphe.

- Attribuer un identificateur de session. La fenêtre principale du prototype offre un champ de texte pour saisir la référence du graphe. Cette référence sera aussi enregistrée dans le fichier XML décrivant le graphe de session.

Simulation du déploiement et de la reconfiguration

En partant du graphe conçu dans la zone de gauche, on peut choisir de simuler le déploiement ou la reconfiguration. Dans la suite, nous allons décrire ces deux procédures :

Simulation du Déploiement

Un participant qui entre dans la session, est simulé au niveau du prototype par l'ajout d'un nœud au graphe de simulation. Ainsi, en cliquant sur le bouton  , une boite de dialogue s'affiche. Elle permet de choisir un nœud et de lui associer un contexte d'exécution. Le prototype procède alors aux différentes étapes de l'algorithme du déploiement.

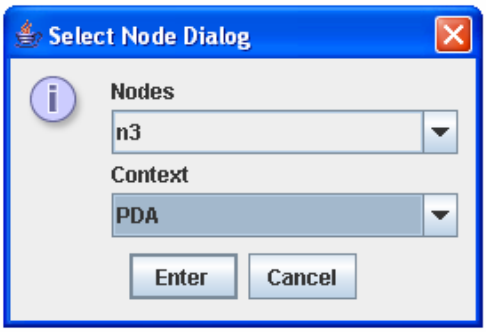


Figure 5.6. Attribution d'un contexte d'exécution au nœud de déploiement

Après le déploiement, une couleur est attribuée au nœud inséré dans le graphe ainsi qu'aux arcs associés (tableau 5.1) :

Couleur	Nœud	Arc
Vert	Le déploiement s'est terminé avec succès.	Les deux nœuds en extrémité de l'arc ont réussi le déploiement.
Jaune	Seulement une partie des applications requises a été déployée.	Seulement un nœud a réussi le déploiement.
Rouge	Echec total du déploiement.	Echec du déploiement des deux côtés de l'arc

Tableau 5.1. Couleurs des nœuds et des arcs

Cette opération peut être répétée tant qu'il reste encore des nœuds à ajouter dans le graphe de simulation. On peut également supprimer un nœud puis le réinsérer à nouveau. De cette façon, on peut simuler différentes situations de connexion et de déconnexion à la session.

La zone dédiée à la simulation offre aussi des boutons permettant de réaliser des actions comme l'agrandissement et la diminution de la taille du graphe.

Simulation de la reconfiguration

La reconfiguration intervient lorsque la session évolue d'un schéma de collaboration vers un autre. Dans le simulateur, cette évolution se traduit par le passage d'un graphe de session à un autre graphe. Ainsi, après le déploiement initial, l'administrateur peut appliquer des changements au niveau du graphe de conception en ajoutant et en supprimant des arcs entre les nœuds.

Dans la fenêtre principale du simulateur, on aura deux graphes : celui de gauche donne le nouvel état de la session, celui de droite donne l'état précédent de la session dont on a réalisé le déploiement. En cliquant sur le bouton  , la reconfiguration est démarrée. L'algorithme reçoit alors le nouveau graphe, le compare avec le graphe sur lequel on a réalise le déploiement. Ensuite, il procède aux différentes étapes de la reconfiguration comme décrit dans la section 3.4.3.

La simulation du déploiement et de la reconfiguration se base principalement sur le module contrôle du déploiement de la couche déploiement. En particulier, *DeploymentEngineSimulation* est utilisé pour le déploiement. *PreDynamicDeploymentSimulation* et *DynamicDeploymentSimulation* sont utilisés pour la reconfiguration.

Visualisation des résultats

Le récapitulatif des étapes du déploiement et de la reconfiguration peut être visualisé dans une fenêtre à part. Cette fenêtre offre cinq zones que nous détaillons dans la suite :

- **Nœuds**. En utilisant le composant JTree de Java, les nœuds ayant été insérés dans le graphe de simulation seront affichés avec chacun le descriptif de contexte d'exécution correspondant.

- **Besoins**. Cette zone permet d'afficher les besoins de chaque nœud inséré dans le graphe de simulation. Un besoin est décrit sous la forme d'un arbre. Le premier niveau détaille les éléments suivants : l'état du déploiement (succès ou échec), les nœuds voisins, les applications découvertes, les flux de données, et les implantations candidates. De la même façon, on utilise le composant JTree pour représenter cet arbre.

- **Implantations déployées pour chaque besoin.** En sélectionnant un besoin, cette zone permet d'énumérer les implantations qui ont été choisies pour couvrir la solution. Il utilise pour cela une structure en arbre. Chaque implantation est décrite par trois éléments : son descripteur, les voisins interopérables, et le nombre d'instantiation permettant de satisfaire les nœuds voisins.

- **Implantations déployées pour chaque nœud.** A la différence de la zone précédente, on décrit ici les implantations en prenant en compte tous les besoins du nœud

sélectionné. Pour cela, on ajoute un élément indiquant les besoins couverts. Le nombre d'instanciation ainsi que les voisins concernés, ne se limite pas à un seul besoin.

- **Configurations.** Les configurations de déploiements générées pour un besoin particulier, sont affichées dans une table. Chaque ligne renseigne sur le numéro de la configuration, son poids, son état (valide ou non), et les implantations qui forment la configuration.

Scénario de simulation du déploiement et de la reconfiguration

Nous allons décrire le fonctionnement du simulateur du déploiement à travers un exemple illustratif. Pour cela, nous considérons une session formée par 4 participants. Nous allons d'abord simuler l'entrée de ces participants dans la session. Puis, nous allons considérer différentes situations possibles d'évolution de la session et nous allons simuler la reconfiguration pour chaque cas.

Simulation du déploiement

La figure 5.7 donne le graphe décrivant le modèle de session que nous voulons tester. Les nœuds doivent avoir les applications adéquates pour permettre les échanges des flux textuels entre les participants de la session. Par exemple, le nœud n1 doit déployer une application lui permettant d'envoyer le même flux textuel t1 vers les nœuds {n2, n3, et n4}.

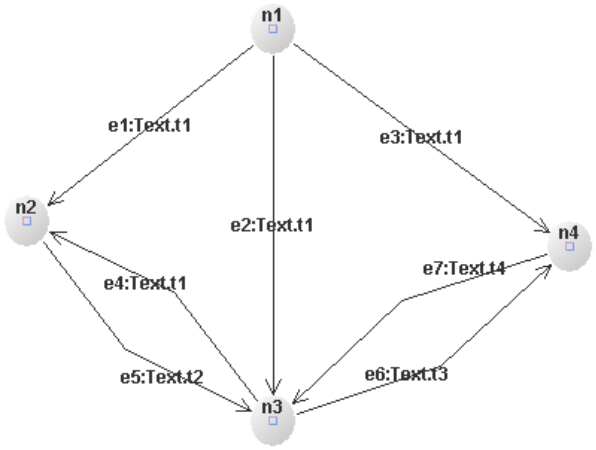


Figure 5.7. Modèle de session

Nous supposons qu'au départ de la simulation, la session est vide. L'étape 1 démarre lorsque l'on simule l'entrée du nœud n1. L'algorithme de déploiement exécute les opérations de déploiement en cherchant localement les descripteurs des applications permettant de produire un flux textuel. Remarquez que la phase de vérification de l'interopérabilité retourne vrai puisqu'il n'y a pas d'autre nœud.

L'étape 2 démarre avec la simulation de l'entrée du nœud n2. Dans cette étape, nous avons supposé qu'il n'y pas d'application. Pour cette raison, l'algorithme de déploiement ne trouve aucune configuration de déploiement couvrant les besoins du nœud n2. C'est pourquoi la couleur du nœud du graphe devient rouge et l'arc entre n1 et n2 est jaune.

Ensuite, dans la troisième étape, nous avons supposé que les applications disponibles permettent la production et la consommation des flux textuels. A la fin de la simulation, le nœud n3 est vert ce qui signifie le succès du processus de déploiement.

Enfin, pour la dernière étape, nous avons simulé l'entrée du nœud n4 en présence seulement d'applications permettant la production de flux textuels. Le résultat du déploiement correspond parfaitement à nos attentes puisque l'algorithme de déploiement a trouvé la configuration de déploiement couvrant le besoin de production. Quant au besoin de consommation, l'état des applications disponibles ne permet pas de trouver une configuration de déploiement adéquate.

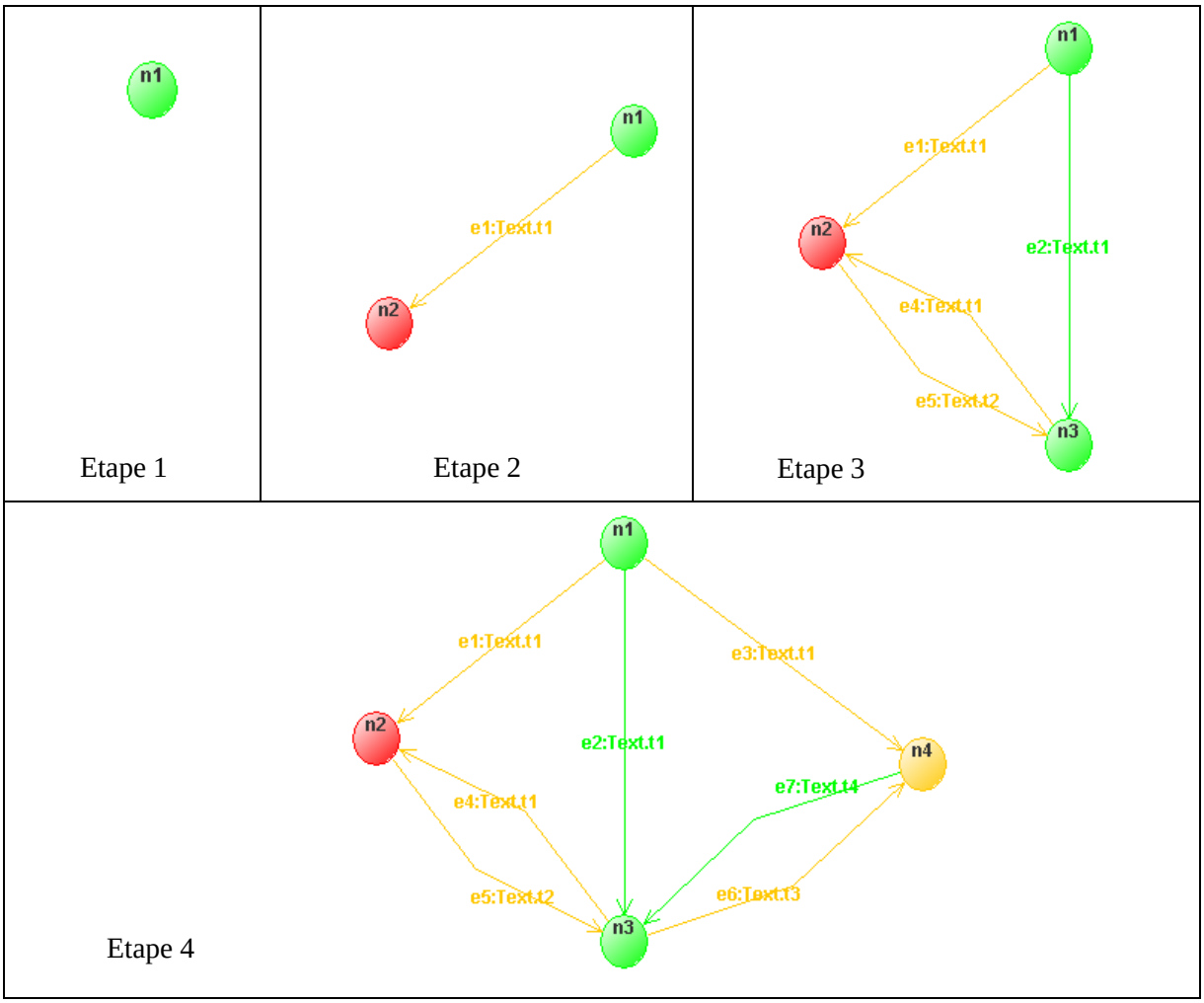


Figure 5.8. Simulation du déploiement initial

Simulation de la reconfiguration

Afin de valider le processus de reconfiguration, nous avons repris le résultat du déploiement initial caractérisé par deux nœuds (n1 et n3) ayant réussi totalement le déploiement, un nœud (n4) ayant réussi partiellement le déploiement et un nœud (n2) en attente des applications adéquates. Nous avons supposé que les applications disponibles permettent la production et la consommation des flux textuels et nous avons déclenché la reconfiguration.

Comme nous l'avons décrit précédemment, l'algorithme commence d'abord par découvrir les nœuds qui nécessitent la reconfiguration. Dans notre cas, il s'agit des nœuds n2 et n4. Ensuite l'algorithme exécute le reste des opérations de déploiement. Le résultat est affiché dans la figure 5.9.

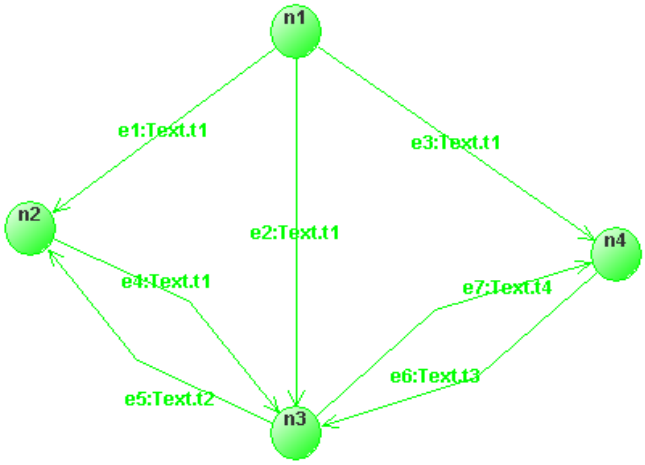


Figure 5.9. Attribution d'un contexte d'exécution au nœud de déploiement

Dans le prototype, les opérations de découverte et de rapatriement sur le réseau Pair-à-Pair, ont été simulées par des accès sur le disque local. Ceci nous a permis d'avoir une première implantation de l'algorithme de déploiement et de reconfiguration qui génère correctement les configurations de déploiement. Grâce à ces résultats, il était possible de continuer les développements en se focalisant cette fois sur les communications réelles entre les nœuds.

5.2.2 Sessions collaboratives synchrones

Dans le cadre de l'utilisation de notre API dans un environnement réel, nous avons développé quatre autres prototypes dont le but est de démontrer la flexibilité de notre API. Ces prototypes sont :

- Prototype Fournisseur d'applications,
- Prototype Demandeur d'applications,
- Prototype Fournisseur et demandeur d'applications,
- Prototype de visualisation de l'état global du déploiement.

La figure 5.10 représente une topologie possible de la session. Les prototypes développés sont décrits ci-dessous.

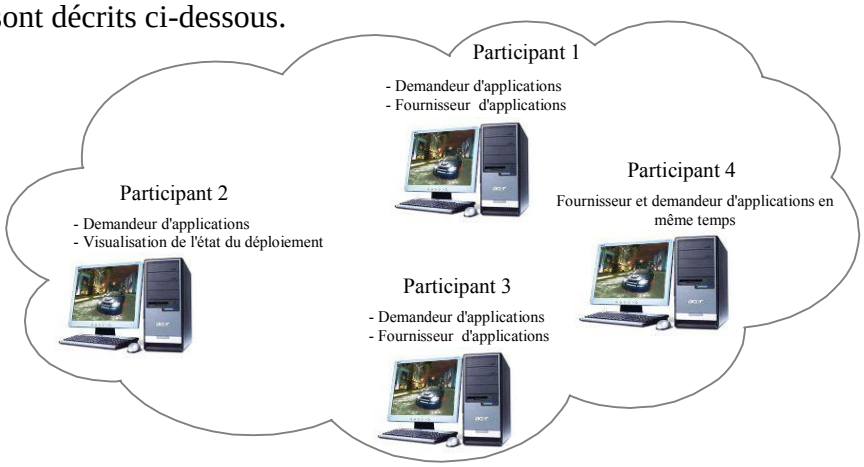


Figure 5.10. Application à la cohérence de sessions collaboratives

Prototype Fournisseur d'applications

Le prototype Fournisseur d'applications (figure 5.11) permet à un nœud d'héberger des applications et de délivrer à la demande les implantations demandées. Généralement, il est

installé sur le nœud le plus stable et le plus accessible. Ce prototype utilise les modules *Gestion du réseau P2P*, *Gestion des applications*, et *Gestion distribuée des applications* de la couche déploiement. L'objectif de ce prototype est de montrer qu'il est possible de créer plusieurs processus serveurs et d'attacher à chaque processus un ou plusieurs dépositaires contenant les contenus à publier.

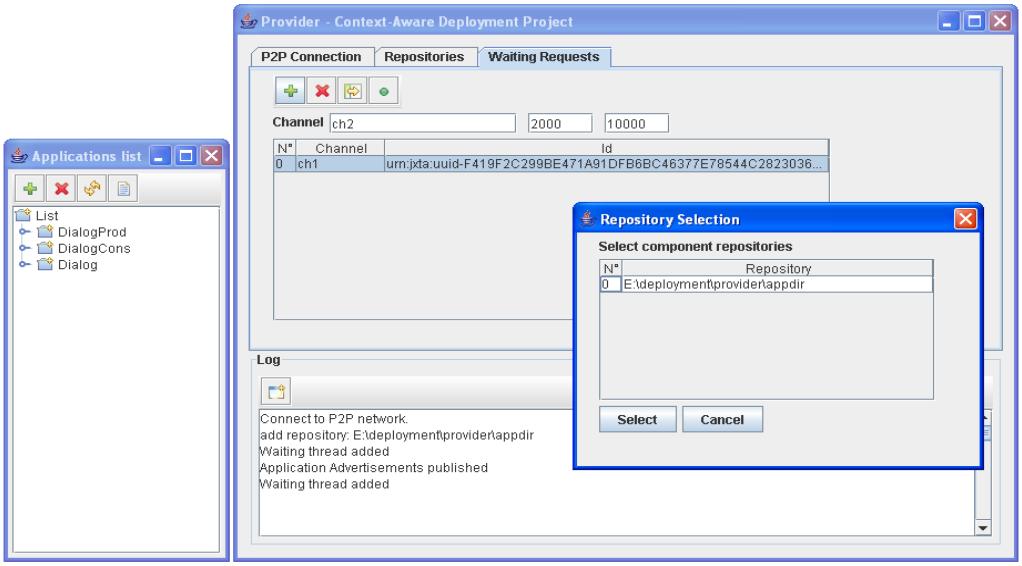


Figure 5.11. Fenêtre principale du prototype fournisseur d'applications

La fenêtre principale de ce prototype offre trois onglets :

Accès au réseau P2P

Le fournisseur peut rejoindre le groupe de pairs et être dans le même espace que les membres de la session collaborative en utilisant cet onglet. Pour cela, il faut remplir les champs indiquant l'identité du nœud, le nom d'utilisateur, un mot de passe, et le nom de la session. Ensuite, il faut cliquer sur le bouton  pour démarrer l'application P2P.

Gestionnaire d'application

Cet onglet permet de choisir les dépositaires qui contiennent les contenus des applications à publier. Pour chaque dépositaire, une ligne sera ajoutée dans la table affichée dans l'onglet. En sélectionnant une ligne et en cliquant sur le bouton  , les contenus du dépositaire correspondant seront affichés dans une nouvelle fenêtre sous forme d'arbre. Avec  , il est possible aussi de supprimer une ligne de la table sans supprimer réellement les contenus du disque dur.

Gestionnaire des processus serveurs

A travers cet onglet, le fournisseur d'application peut démarrer plusieurs processus à l'écoute des demandes de rapatriement. Les paramètres nécessaires pour initialiser un processus serveur sont : le nom du canal de communication, le délai maximal pour la découverte de l'annonce du canal et le délai maximal qui spécifie la durée au delà de laquelle la connexion déjà ouverte sera annulée s'il n'y a pas une requête du côté du demandeur. En ajoutant le processus avec le bouton  , une boite de dialogue s'affiche indiquant les dépositaires à sélectionner pour les associer au serveur. En sélectionnant la ligne sur la table des processus et en cliquant sur le bouton  , le fournisseur parcourt les dépositaires,

construit des annonces d'applications et les publie dans son cache. Le bouton permet d'afficher une fenêtre contenant toutes les annonces d'applications et de canaux disponibles localement.

Prototype Demandeur d'applications

Un nœud exécutant ce prototype remplit le rôle d'un nœud de déploiement sur lequel l'algorithme de déploiement va pouvoir générer des configurations de déploiement et ainsi installer les applications qui manquent. Le prototype (figure 5.12) utilise les modules gestion du réseau P2P, contrôle du déploiement, gestion des implantations déployées, gestion distribuée des implantations déployées, et gestion du contexte local.

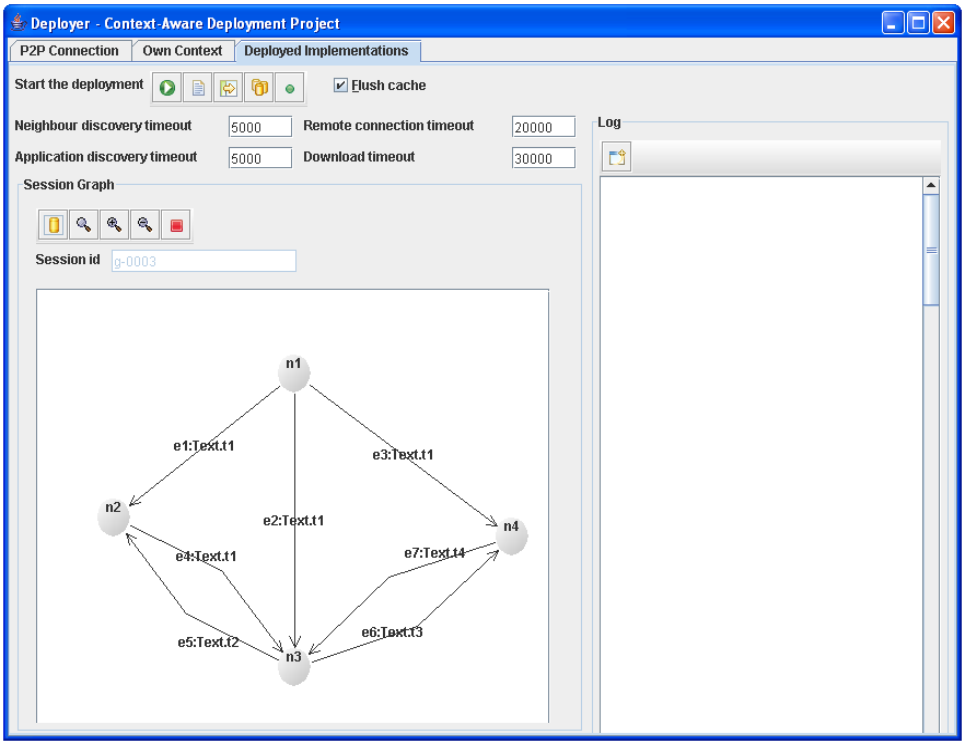


Figure 5.12. Fenêtre principale du prototype demandeur d'applications

La fenêtre principale offre trois onglets :

Accès au réseau P2P

Comme pour le fournisseur, cet onglet permet de rejoindre le groupe de pairs en saisissant d'abord l'identité du nœud, le nom de l'utilisateur, le mot de passe, et le nom de la session. Pour ne pas chercher indéfiniment l'annonce du groupe de pairs, la durée maximale de la découverte doit être saisie dans l'endroit approprié.

Gestionnaire des descripteurs locaux

Cet onglet offre deux zones permettant de visualiser d'un côté le descripteur de contexte d'exécution et de l'autre côté le descripteur des implantations déployées. Ces deux descripteurs sont affichés sous forme d'arbre. Afin de publier le contenu décrivant les implantations déployées, le nœud de déploiement démarre un processus serveur en écoute des requêtes de rapatriement. Les paramètres nécessaires à saisir sont le nom du canal de communication, la durée maximale pour la découverte de l'annonce du canal et le délai

maximal qui spécifie la durée au delà de laquelle s'il n'y a pas une requête du côté du demandeur, la connexion déjà ouverte sera annulée.

Gestionnaire de déploiement

Cet onglet permet à un membre de la session de démarrer le processus de déploiement. Ainsi, le participant ouvre un fichier XML décrivant le graphe de la session pour laquelle on veut réaliser le déploiement. Ce fichier va être parcouru pour construire le graphe correspondant et l'afficher dans un canevas approprié. L'onglet permet aussi de saisir des paramètres nécessaires pour assurer le critère de complétude du déploiement :

- Durée maximale de la découverte des contenus des applications ;
- Durée maximale de la découverte des contenus décrivant les implantations déployées sur les nœuds voisins ;
- Durée au delà de laquelle de si le fournisseur ne donne pas de signe, la connexion est annulée ;
- Durée maximale d'attente de la réponse de rapatriement du fournisseur.

En cliquant sur le bouton  , le processus de déploiement est démarré. Il faut que l'identité du nœud de déploiement corresponde à l'identité d'un nœud du graphe. Une fois terminé, le prototype attribue une couleur au nœud de déploiement, à ses voisins directs, ainsi qu'aux arcs entre ces nœuds. Dans le cas où le prototype ne reçoit pas d'information sur un nœud voisin, alors sa couleur ne change pas et reste grise.

Après le rapatriement des implantations, le participant clique sur le bouton  . Cette action permet d'une part de créer et d'enregistrer le contenu décrivant les implantations déployées, et d'autre part de créer et de publier l'annonce de ce contenu pour que les autres nœuds puissent la découvrir. Toutes les annonces dans le cache peuvent être visualisées en cliquant sur le bouton  .

Le résultat du déploiement peut être affiché dans une fenêtre séparée. Cette fenêtre permet en particulier de sélectionner une implantation déployée et de l'exécuter. Le nombre d'instance dépend du résultat du déploiement.

Prototype Fournisseur et demandeur en même temps

Le nœud possède un double rôle, celui de demandeur et de fournisseur à la fois. Il combine tous les modules de la couche déploiement. On retrouve donc les onglets suivants (figure 5.13) : accès au réseau P2P, gestionnaire des descripteurs locaux, gestionnaire des applications, gestionnaire des processus serveurs, et contrôle du déploiement.

Lorsque le déploiement est terminé, les contenus des implantations déployées sont convertis en contenus d'applications et sont enregistrés dans le dépositaire prévu pour être parcouru par le processus serveur en cas de demande. Ces contenus sont aussi publiés dans le cache après avoir créé les annonces correspondantes. Ainsi, le nœud de déploiement devient aussi un fournisseur d'applications que les autres nœuds peuvent interroger.

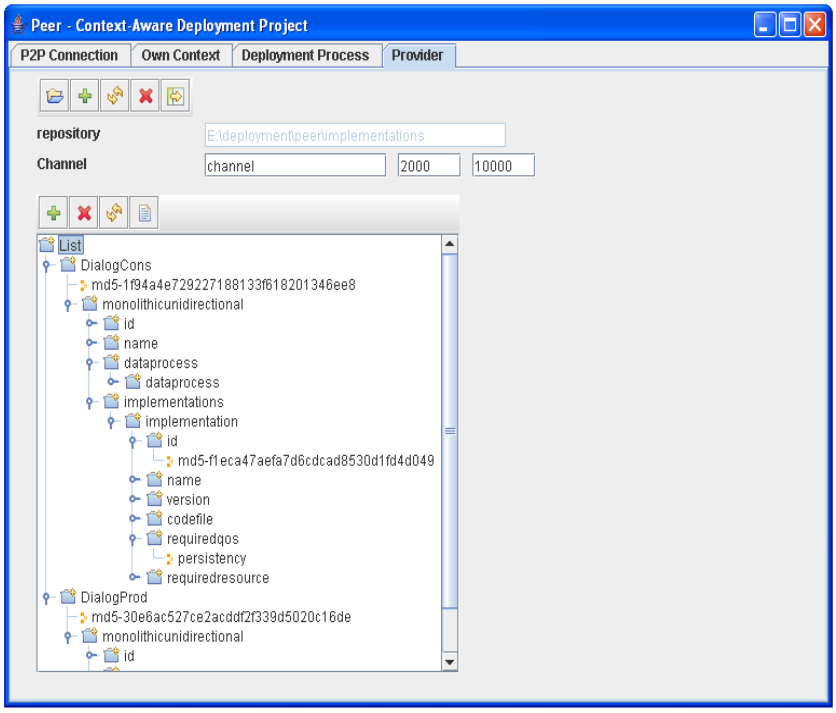


Figure 5.13. Fenêtre principale du prototype fournisseur / demandeur d'applications

Prototype de visualisation de l'état du déploiement dans la session

Ce prototype (figure 5.14) permet d'avoir une vue globale sur l'état du déploiement des nœuds formant la session en cours. Le principe est de découvrir, puis de rapatrier les contenus des descripteurs décrivant les implantations déployées sur chaque nœud de la session. A partir de ces contenus, le prototype attribue une couleur à chaque nœud du graphe modélisant la session ainsi qu'aux arcs entre les nœuds. Il offre également une fenêtre secondaire permettant de rejoindre le groupe de pairs et ainsi être dans le même espace que les membres de la session. Il utilise essentiellement les modules gestion du réseau P2P et gestion distribuée des implantations déployées.

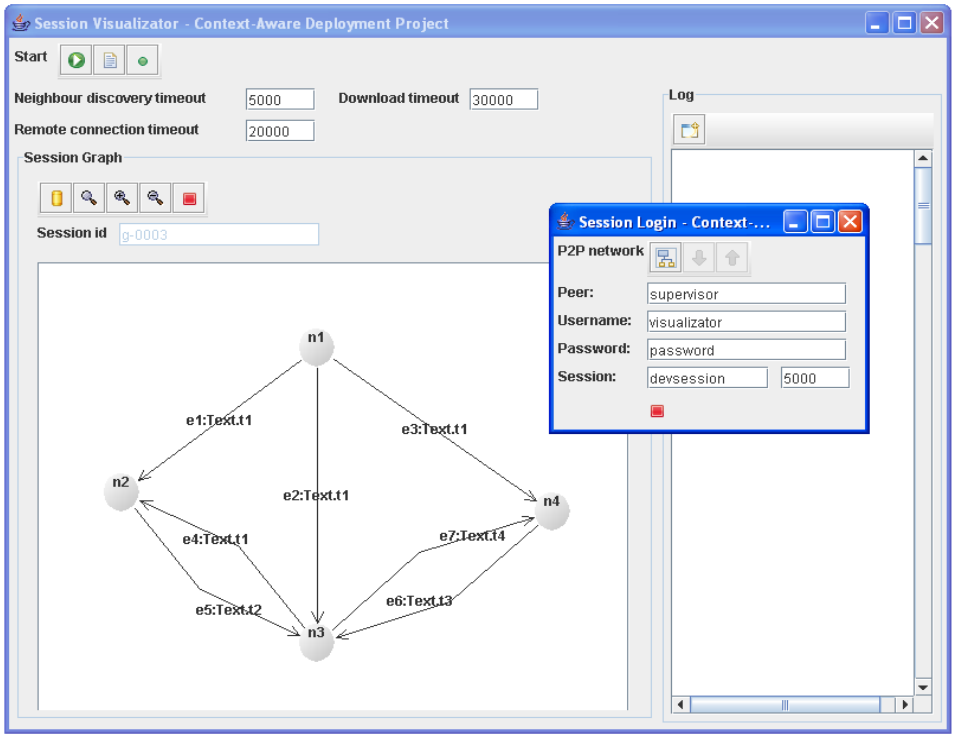


Figure 5.14. Fenêtre principale du prototype pour visualiser l'état du déploiement

5.3 Evaluation de l'algorithme de déploiement

5.3.1 Configuration de l'expérimentation

Le déploiement des applications sur le nœud d'un membre de la session passe par un ensemble d'étapes. Les plus importantes sont :

- La vérification de la compatibilité des implantations candidates avec l'environnement d'exécution qu'offre le nœud de déploiement (contexte local) ;
- La vérification de l'interopérabilité avec les implantations déployées sur les nœuds voisins (contexte global) ;
- La génération des configurations de déploiement à partir des implantations sélectionnées ;
- Et enfin, la validation de ces configurations permettant de ne retenir que celles qui couvrent les besoins du participant à la session.

L'objectif des expérimentations est d'évaluer les performances de l'algorithme de déploiement. Pour cela, on calcule le temps d'exécution nécessaire pour accomplir chaque étape tout en faisant varier le nombre d'implantations et le nombre de nœuds voisins.

Cette étude se focalise sur la couche déploiement de notre architecture, en particulier le module contrôle du déploiement. Pour cette raison, nous avons mené les expérimentations indépendamment du réseau Pair-à-Pair. Nous avons donc introduit quelques modifications au niveau du prototype simulateur de déploiement nous permettant de :

- Générer automatiquement des contenus d'application. Chaque application contient une seule implantation. Le nombre de contenus est fixé lors de l'exécution du prototype.
- Générer automatiquement des graphes de sessions en faisant varier le nombre de voisins du nœud principal. Ainsi, les tests de déploiement sont réalisés pour quatre schémas de coopération : (1) Un nœud principal et seulement un nœud voisin. (2) Un nœud principal et dix nœuds voisins. (3) Un nœud principal et cent nœud voisins. (4) un nœud principal et deux cents nœuds voisins. Nous supposons que les nœuds voisins ont réalisé le déploiement et nous simulons l'entrée du nœud principal. Les mesures de déploiement sont prises pour ce nœud.
- Réaliser des itérations du processus de déploiement, afin de relever plusieurs mesures concernant le déploiement sur le nœud principal. Ces mesures sont enregistrées dans des fichiers appropriés. Le nombre de répétition est fixé lors du démarrage du prototype.

Pour faire les tests de déploiement, le simulateur a été placé sur une machine ayant les caractéristiques suivantes :

- Système d'exploitation : Windows XP Professionnel
- Type et fréquence du processeur : Pentium ® 4 avec 3.20 GHz
- Taille de la mémoire : 1.00 Go
- Version de la machine virtuelle Java : Sun JVM V1.4.1_05

Dans la suite, nous allons monter les résultats de ces mesures qui sont cohérentes avec nos attentes et nos hypothèses.

5.3.2 Vérification de la compatibilité

Dans cette expérimentation, nous avons pour objectif de mesurer le temps d'exécution nécessaire à un nœud pour réaliser la phase de vérification de la compatibilité:

Résultats de l'expérimentation

Le tableau 5.2 montre la variation du temps d'exécution nécessaire à la vérification de la compatibilité en fonction du nombre d'implantation candidates et du nombre de nœuds voisins. Chaque ligne du tableau (N1, N10, N100 et N200) donne le nombre de nœuds voisins du nœud principal (nombre respectivement égal à 1, 10, 100 et 200). Chaque colonne donne le nombre d'implantations candidates (de 100 à 700 implantations). Les valeurs de chaque case donnent le temps d'exécution en ms pour la vérification de la compatibilité. Pour chaque valeur, nous avons répété les mesures 1000 fois et nous avons calculé la moyenne. La figure 5.15 représente graphiquement les résultats du tableau 5.2.

	100	200	300	400	500	600	700
N1	12,19	21,97	33,41	42,25	52,61	65,93	108,79
N10	9,83	23,73	34,84	45,21	51,73	66,32	109,81
N100	17,88	21,88	37,77	47,33	54,57	66,08	103,05
N200	9,75	25,78	32,64	44,37	55,49	65,08	109,02

Tableau 5.2. Temps d'exécution en ms pour la vérification de la compatibilité

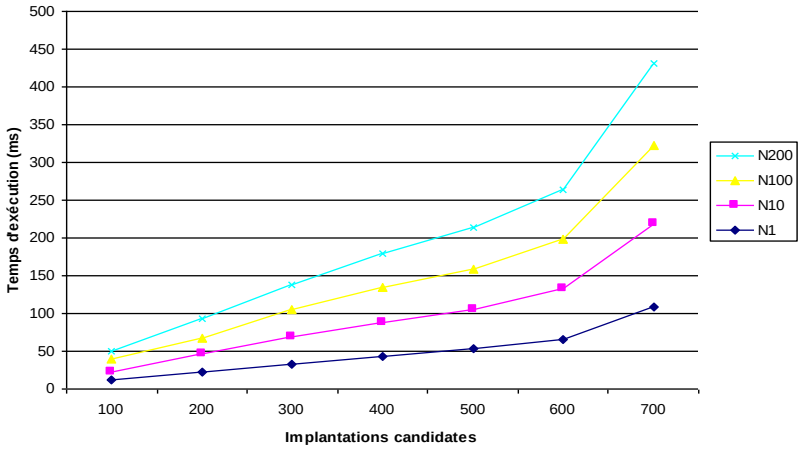


Figure 5.15. Vérification de la compatibilité

Discussion des résultats de l'expérimentation

En termes de fonctionnalité, la procédure de vérification de la compatibilité a réussi son objectif. A partir d'un ensemble d'implantations candidates, elle peut les filtrer et ne garder que celles qui sont compatibles avec l'environnement d'exécution.

D'après la courbe, on remarque que les temps d'exécution suivent une évolution linéaire jusqu'au nombre 600 implantations. Puis, ils augmentent de façon exponentielle au delà de cette valeur.

Ces résultats ont montré que le nombre d'implantations n'influe pas sur le temps unitaire nécessaire pour vérifier la compatibilité d'une implantation. Jusqu'à 600 implantations, ce temps unitaire a une valeur très petite, de l'ordre de 0,1 ms. Cette petite

valeur explique que les temps obtenus restent raisonnables pour traiter quelques centaines d'applications.

Les valeurs obtenues pour la vérification de la compatibilité restent très proches les une des autres, lorsque l'on fait varier le nombre des nœuds voisins de un jusqu'à deux cents. Ceci est justifiable puisque la vérification de la compatibilité concerne l'implantation avec son environnement d'exécution, indépendamment des voisins et de leur nombre.

5.3.3 Vérification de l'interopérabilité

Dans cette expérimentation, nous avons pour objectif de mesurer le temps d'exécution nécessaire à un nœud pour réaliser la phase de vérification de l'interopérabilité.

Résultats de l'expérimentation

Les tableaux 5.3 ainsi que la courbe de la figure 5.16 illustrent les résultats des mesures du temps d'exécution nécessaire pour vérifier l'interopérabilité. Comme pour le tableau précédent, chaque ligne (N1, N10, N100 et N200) donne le nombre de nœuds voisins du nœud principal (nombre respectivement égal à 1, 10, 100 et 200) et chaque colonne donne le nombre d'implantations candidates (de 100 à 700 implantations). Les valeurs de chaque case donnent le temps d'exécution en ms pour la vérification de l'interopérabilité.

	100	200	300	400	500	600	700
N1	1,09	1,87	3,26	3,91	16,56	3,26	13,74
N10	8,94	20,65	22,93	26,55	131,05	39,98	101,77
N100	122,8	253,45	370,99	463,26	563,62	676,98	847,03
N200	362,85	783	1119,1	1449,42	1877,2	2313,15	2582,23

Tableau 5.3. Temps d'exécution en ms pour la vérification de l'interopérabilité

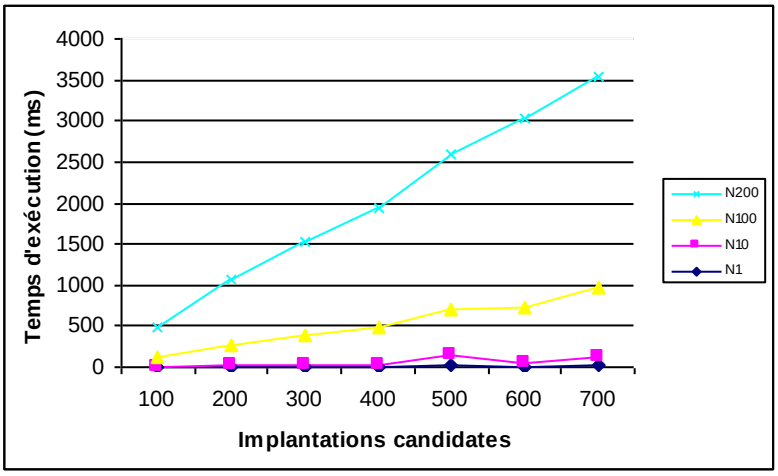


Figure 5.16. Vérification de l'interopérabilité

Discussion des résultats de l'expérimentation

La première remarque est que la phase de vérification de l'interopérabilité a réussi son objectif malgré les nombres croissants des implantations candidates et des nœuds voisins.

Lorsque l'on considère un nombre constant de nœuds voisins, le temps d'exécution suit une évolution linéaire en fonction du nombre d'implantations. Ainsi le temps unitaire nécessaire pour traiter une implantation est constant. Pour un seul nœud voisin ainsi que pour dix nœuds, ce temps unitaire nécessaire pour vérifier l'interopérabilité est négligeable, de

l'ordre d'une dizaine de microsecondes. Par contre pour cents nœuds voisins, ce temps est de l'ordre de 1,1 ms. Enfin pour deux cents nœuds voisins, la vérification de l'interopérabilité nécessite environ 3,9 ms par application.

La vérification de l'interopérabilité dépend directement du nombre de voisins liés au nœud de déploiement. Ceci est montré par la pente de chaque courbe de la figure 4.16. Cette pente est d'autant plus forte que le nombre de voisins est élevé. Plus le nombre de nœuds voisins augmente, plus le temps unitaire nécessaire par application augmente. De ce fait, 100 voisins par nœud nous semble la limite supérieure à ne pas dépasser pour ne pas avoir des délais trop longs.

5.3.4 Génération des configurations de déploiement

Dans cette expérimentation, nous avons pour objectif de mesurer le temps d'exécution nécessaire à un nœud pour réaliser la phase de génération des configurations de déploiement.

Résultats de l'expérimentation

Les mesures concernant la génération des configurations de déploiement sont illustrées par le tableau 5.4. Comme pour les tableaux précédents, chaque ligne (N1, N10, N100 et N200) donne le nombre de nœuds voisins du nœud principal (nombre respectivement égal à 1, 10, 100 et 200) et chaque colonne donne le nombre d'implantations candidates (de 5 à 19 implantations). Les valeurs de chaque case donnent le temps d'exécution en ms pour la génération des configurations de déploiement.

Ces mesures sont reproduites par les courbes de la figure 5.17. Pour limiter l'effet exponentiel lors de l'affichage, nous avons choisi une échelle logarithmique pour représenter les temps d'exécution.

	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
N1	0,94	0,16	3,44	0,16	3,13	2,35	7,64	30,61	115,06	446,12	1745,83	6985,49	38114,1	140531,1	628845,1
N10	0,32	0,16	0,78	0,15	1,1	3,41	7,99	27,91	112,66	446,35	1761,26	7050,04	38554,2	143325,9	617991
N100	0,16	0,16	0,48	0	0,63	1,89	7,31	26,37	108,62	440,35	1795,98	7175,94	36442,6	147468,1	661495,2
N200	0,15	0,46	0,48	0,16	1,39	1,4	7,18	23,81	109,16	442,42	1812,09	7709,9	38148,1	157029,7	697046,7

Tableau 5.4. Temps d'exécution en ms pour la génération des configurations de déploiement

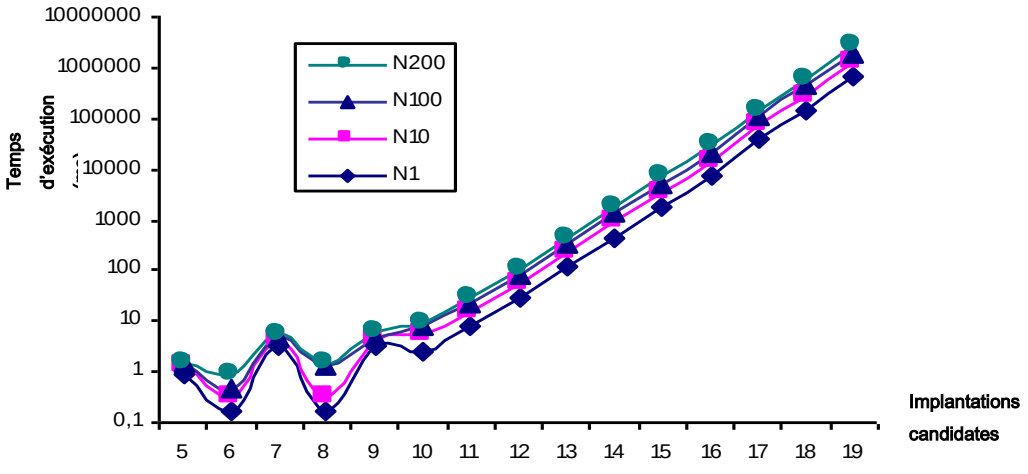


Figure 5.17. Génération des configurations de déploiement (échelle logarithmique)

Discussion des résultats de l'expérimentation

Lorsque l'on considère un nombre constant d'implantations et que l'on fait varier le nombre de nœuds voisins, on remarque que les valeurs obtenues sont très proches les unes des autres. Elles ne dépendent donc pas du nombre des nœuds voisins. En effet, la génération de configuration s'occupe de trouver toutes les combinaisons possibles à partir d'un certain nombre d'implantations.

Par contre, on remarque bien d'après la courbe, que la génération des configurations de déploiement suit une évolution fortement exponentielle. On passe de quelques ms pour générer des configurations de déploiement avec 5 implantations jusqu'à environ 3 h pour générer des configurations de déploiement avec 19 implantations. De ce fait, la phase de génération des configurations de déploiement de notre algorithme ne peut pas traiter plus de 19 implantations à la fois.

5.3.5 Validation des configurations de déploiement

Dans cette expérimentation, nous avons pour objectif de mesurer temps d'exécution nécessaire à un nœud pour la phase de validation des configurations de déploiement.

Résultats de l'expérimentation

Le tableau 5.5 suivant ainsi que la figure 5.18 illustrent les résultats des mesures de l'étape de validation des configurations de déploiement. Comme pour les tableaux précédents, chaque ligne (N1, N10, N100 et N200) donne le nombre de nœuds voisins du nœud principal (nombre respectivement égal à 1, 10, 100 et 200) et chaque colonne donne le nombre d'implantations candidates (de 5 à 19 implantations). Les valeurs de chaque case donnent le temps d'exécution en ms pour la validation des configurations de déploiement.

Comme précédemment, pour la figure 5.18, nous avons choisi une échelle logarithmique pour représenter les temps d'exécution, afin de limiter l'effet exponentiel lors de l'affichage.

	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
N1	0,16	0	0,31	0,94	0,79	1,41	5,62	6,86	12,91	13,3	19,31	32,95	78,79	150,18	6774,61
N10	0,32	0	0,47	1,09	1,56	5,49	11,39	16,28	25,14	43,98	74,04	136,68	280,42	559,97	12387
N100	1,86	2,59	7	16,13	38,33	63,53	116,57	232,06	456,52	896,86	1788,25	3610,71	7099,94	14272,3	28530,1
N200	5,92	14,25	26,54	50,2	107,82	203,24	403,01	798,2	1608,5	3123,6	6284,19	12701	24989,5	50203,3	101217,8

Tableau 5.5. Temps d'exécution en ms pour la validation des configurations de déploiement

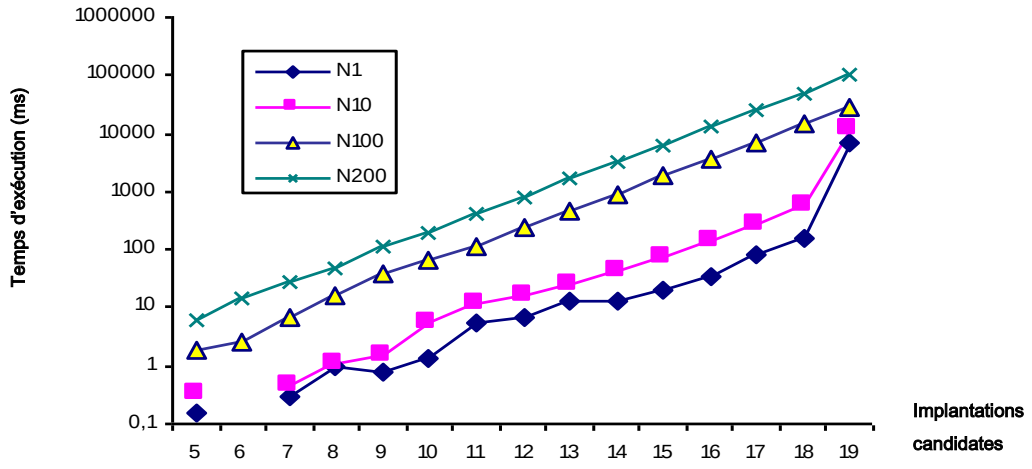


Figure 5.18. Validation des configurations de déploiement (échelle logarithmique)

Discussion des résultats de l'expérimentation

La procédure de validation des configurations de déploiement se déroule correctement en faisant varier les nombres d'implantations et les nombres de voisins.

La figure 5.18 montre que le nombre de voisins influence de façon linéaire la durée de la phase de validation. Ceci est normal car cette phase fait directement appel aux nœuds voisins et à leurs caractéristiques.

Par contre, lorsque l'on considère un nombre de voisins constant, chaque courbe suit une évolution fortement exponentielle. Les mesures s'arrêtent à 19 implantations puisque l'étape de génération ne peut pas dépasser cette valeur.

5.3.6 Bilan de l'évaluation de déploiement

Le tableau 5.6 résume l'influence du nombre d'implantations et du nombre de voisins selon les quatre étapes précédemment analysées.

Le nombre d'implantations a une forte influence pour les étapes de vérification de la compatibilité, de génération des configurations et de validation des configurations. Il a une faible influence dans l'étape de vérification de l'interopérabilité.

Le nombre de voisins a une forte influence dans l'étape de vérification de l'interopérabilité et une faible influence dans les étapes de vérification de la compatibilité et de validation des configurations.

Etape	Nombre d'implantations	Nombre de voisins
Vérification de la compatibilité	Forte	Faible
Vérification de l'interopérabilité	Faible	Forte
Génération des configurations	Forte	Aucune
Validation des configurations	Forte	Faible

Tableau 5.6. Influence des paramètres nombre d'implantations et nombre de voisins

A partir de ces expérimentations, les limites expérimentales raisonnables pour utiliser l'algorithme de déploiement sur un nœud sont de moins de 100 voisins et de moins de 600 implantations pour les étapes de vérification de la compatibilité et de vérification de l'interopérabilité. Avec leur comportement fortement exponentiel, les étapes de génération des configurations et de validation des configurations ont des valeurs de paramètres beaucoup plus restrictives, limitées à 200 voisins et seulement 13 implantations. Heureusement, comme ces deux dernières étapes interviennent à la fin de l'algorithme de déploiement, elles ne limitent pas aussi fortement les valeurs des entrées de l'algorithme. Le processus de déploiement peut ainsi démarrer avec 100 voisins et 600 applications. Cependant, à la fin de l'étape de vérification de l'interopérabilité, qui filtre et écarte les implantations incompatibles, un maximum de 13 implantations restantes doit être atteint, ces implantations restantes intervenant en entrée de l'étape de génération des configurations. Les valeurs de ces paramètres sont consistantes avec nos observations, observations qui mettent en évidence l'élimination de nombreuses implantations dans les deux premières étapes.

En considérant les limites précédemment identifiées, l'ajout des durées maximales des quatre étapes de l'algorithme de déploiement a une valeur qui reste en dessous de 3.56 secondes. Cette durée, qui semble assez longue, est acceptable du point de vue utilisateur si les entrées et les sorties en session ne sont pas des opérations trop fréquentes. Les sessions de

travail collaboratif synchrone que nous avons envisagées ne sont pas très dynamiques et correspondent au cadre d'application de l'algorithme proposé.

5.4 Evaluation de la couche noyau générique

Les performances de la couche noyau générique ont été évaluées en se basant sur des expérimentations simples mais qui sont réalistes. Notre but est de comprendre et de caractériser certains comportements de la plate-forme JXTA et du réseau sous-jacent qui lie ses composants distribués. Ces expérimentations sont divisées en deux parties :

La première partie se focalise sur la découverte de contenu. Elle fournit des résultats relatifs au service de découverte offert par JXTA. Les tests sur la découverte de contenu sont exécutés selon deux scénarios :

- Local : Les nœuds impliqués dans les expérimentations s'exécutent en local sur une même machine, indépendamment du réseau ;

- Réseau : Les nœuds s'exécutent sur des machines reliées par un réseau LAN à 100 Mb/s.

La deuxième partie se focalise sur le rapatriement de contenu et donne des résultats relatifs aux capacités de communication offertes par JXTA. Ces tests sont uniquement exécutés en local pour ne pas mélanger les temps liés uniquement à JXTA avec ceux liées aux communications réseau.

5.4.1 Evaluation de la découverte des contenus

Dans ce type d'expérimentation, chaque nœud participe au test de découverte en recherchant et en partageant les contenus avec les autres nœuds du réseau Pair-à-Pair. Un nœud publie *n* annonces de contenu. Ainsi, le nombre d'annonces publié est *n* × *taille du groupe de pairs*. Chaque annonce est différente de l'autre. De plus, chaque nœud envoie périodiquement une requête de découverte pour un contenu spécifique choisi aléatoirement et enregistre la réponse. Pour chaque requête, nous mesurons le temps écoulé avant d'obtenir une réponse. Le temps de réponse inclut le temps nécessaire pour transmettre la requête dans le réseau, localiser l'annonce du contenu et retourner la réponse au nœud émetteur. Cette métrique est utilisée pour évaluer l'efficacité de la découverte.

Java fournit la méthode *System.currentTimeMillis()* pour mesurer le temps. La précision de cette méthode est de 16ms sous Windows. Cette approche ne donne pas des mesures suffisamment efficaces pour caractériser des opérations qui nécessitent un temps inférieur à 16ms. Pour résoudre ce problème, nous avons utilisé Java Native Interface (JNI) pour accéder au compteur de haute performance de Windows. Ce dernier permet de donner des valeurs plus précises.

Afin d'évaluer la découverte, nous avons analysé l'impact des facteurs suivants sur la performance : nombre de requêtes émises par chaque nœud, délai entre les requêtes, taille du cache contenant les annonces de contenu, et taille du réseau Pair-à-Pair (seulement pour la configuration réseau). Dans ce qui suit nous présentons les résultats obtenus ainsi que notre analyse.

Configuration locale

Nous avons fait varier le nombre des requêtes envoyées par chaque nœud dans une première plage qui va de 100 à 1000 avec un pas de 100, puis dans une deuxième plage qui va de 1000 à 5000 avec un pas de 1000. En même temps, nous avons fixé le délai entre deux requêtes successives à 1000ms et le nombre des annonces publiés à 100 annonces par nœud. Ce délai de 1000ms est suffisamment grand pour éviter des perturbations entre deux requêtes. Les résultats obtenus sont décrits par les figures suivantes. L'abscisse de chaque courbe représente le nombre *n* de requêtes qu'émet le nœud, l'ordonnée donne le temps de réponse moyen pour chaque requête.

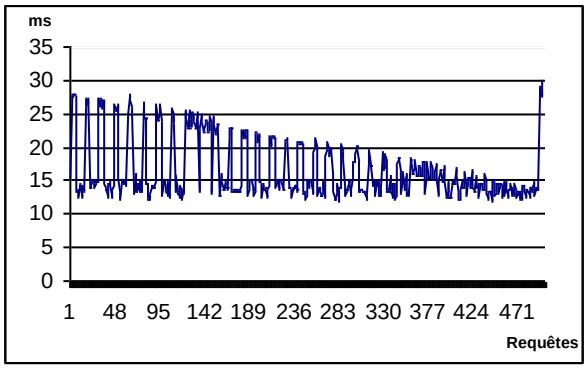


Figure 5.19. Temps de réponse de la découverte de contenu jusqu'à 500 requêtes

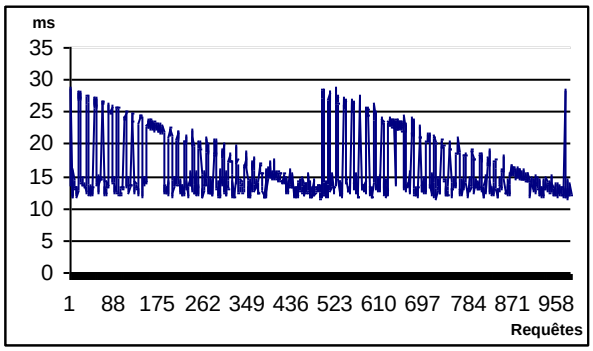


Figure 5.20. Temps de réponse de la découverte de contenu jusqu'à 1000 requêtes

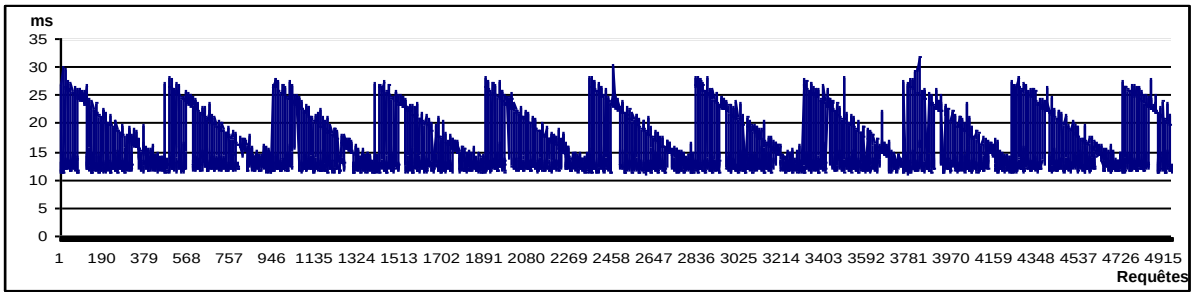


Figure 5.21. Temps de réponse de la découverte de contenu jusqu'à 5000 requêtes

Curieusement, les courbes obtenues montrent le même comportement répétitif chaque 480 requêtes. La durée de chaque période est 480 × 1000 / 60 = 8mn pour tous les tests. Nous remarquons aussi que le temps moyen maximum est de 28ms alors que le temps moyen minimum est de 11,5ms.

Dans le scénario suivant, nous avons fixé le nombre d'annonce des contenus publiés par chaque nœud à 1000 annonces. Nous avons répété les mêmes tests. La fréquence des requêtes a été aussi maintenue à 1 requête par seconde. Les figures donnent les résultats les plus significatifs parmi toutes les expérimentations effectuées. L'abscisse de chaque courbe représente le nombre *n* de requêtes qu'émet le nœud, l'ordonnée donne le temps de réponse moyen pour chaque requête.

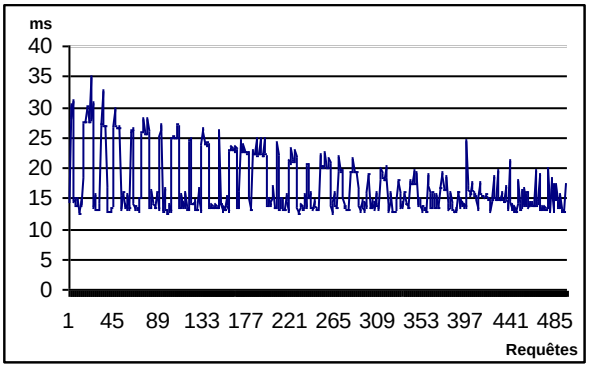


Figure 5.22. Temps de réponse de la découverte de contenu pour 500 requêtes

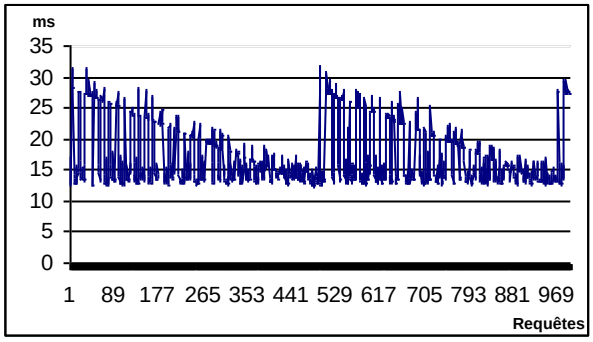


Figure 5.23. Temps de réponse de la découverte de contenu pour 1000 requêtes

Nous avons observé le même comportement répétitif. La période de ce comportement est de 8mn pour tous les tests. Nous notons que le temps moyen maximum est de 30ms alors que le temps moyen minimum est de 13,5ms, ce qui correspond à une augmentation de 2ms par rapport au scénario précédent. Cette variation est relativement mineure lorsqu'on la compare à la variation du nombre d'annonces, dont la valeur est passée de 100 à 1000 annonces.

Dans les expérimentations suivantes, nous avons fixé le nombre de requêtes à 1000 requêtes, le nombre d'annonces à 100 annonces et nous avons varié la fréquence des requêtes de 1, 2, 4, 8, 16, 32 jusqu'à 64 requêtes par seconde. Les résultats les plus significatifs sont montrés par les figures suivantes.

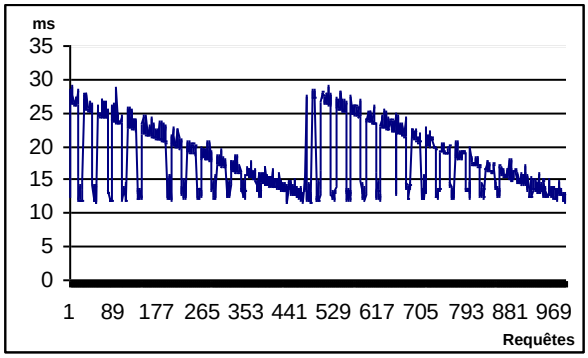


Figure 5.24. Temps de réponse de la découverte de contenu - 1 requête par seconde

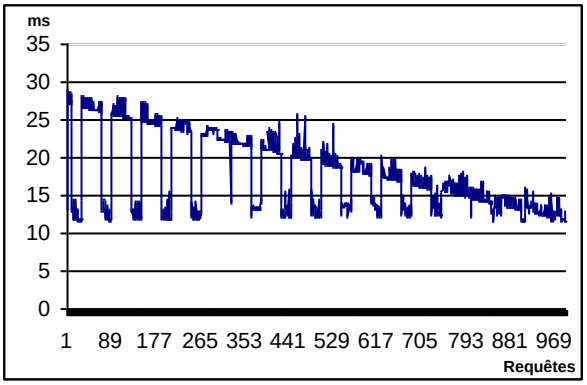


Figure 5.25. Temps de réponse de la découverte de contenu - 2 requêtes par seconde

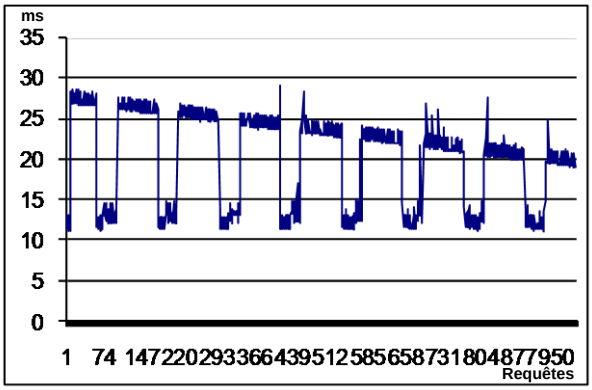


Figure 5.26. Temps de réponse de la découverte de contenu – 4 requêtes par seconde

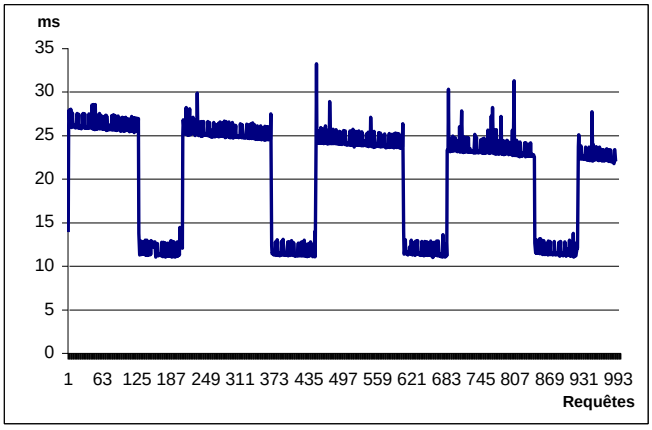


Figure 5.27. Temps de réponse de la découverte de contenu – 8 requêtes par seconde

Les résultats montrent que la durée de chaque période reste à la valeur de 8mn, indépendamment du délai entre les requêtes. Un autre comportement répétitif a été observé. Ce comportement est plus facilement mis en évidence lorsque les requêtes sont émises à une fréquence suffisamment élevée. Il concerne la variation du temps de réponse dans chaque période. Nous pouvons remarquer que le temps de réponse atteint des valeurs hautes pendant 20s puis des valeurs basses pendant 10s pour les tests. Ce comportement est répété périodiquement chaque 30s.

Pour des fréquences élevées (32 et 64 requêtes par seconde), le temps de réponse est très varié. De plus, il est très haut.

Nous allons commenter ces résultats après l'étude de la configuration réseau.

Configuration réseau

L'environnement d'expérimentation consiste en 8 ordinateurs équipés chacun d'un processeur Pentium 4 - 2,80HHz et 1,00Go de mémoire vive. Le système d'exploitation est Windows XP Professionnel pour toutes les machines. Afin d'avoir des résultats significatifs, ces ordinateurs sont reliés par un réseau isolé. La topologie Pair-à-Pair suit une architecture totalement décentralisée appelée *architecture à communication directe* dans la quelle les pairs communiquent directement avec les autres [47].

Nous avons conduit plusieurs expérimentations pour voir l'effet de la charge réseau sur les performances de la découverte. Chaque nœud publie 100 annonces de contenus, annonces générées aléatoirement parmi un ensemble prédéfini de contenus. Nous avons fait varier le nombre de requêtes de découverte envoyé par chaque nœud comme suit : 100, 500, 1000, 5000, et 10000 requêtes. Pour chaque ensemble, nous avons fait varier la fréquence des requêtes de 1, 2, 4, 8, 16, 32, et 64 requêtes par seconde. Chaque scénario a été répété plusieurs fois. Les figures suivantes montrent le temps de réponse pour des fréquences de requêtes égales à 2, 4 et 8 requêtes par seconde.

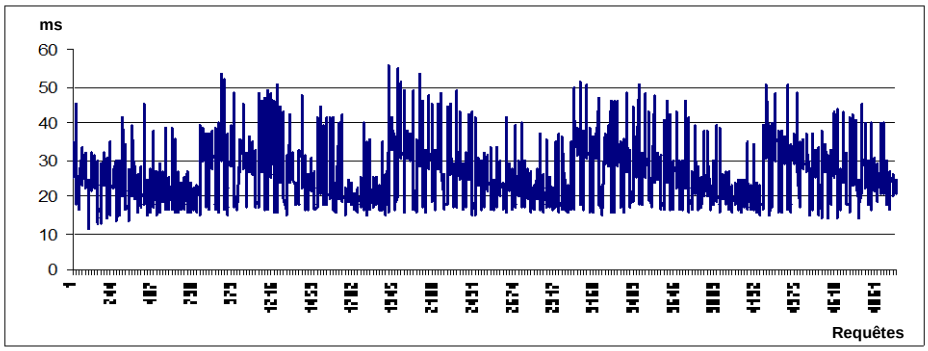


Figure 5.28. Temps de réponse de la découverte de contenu - 2 requêtes par seconde

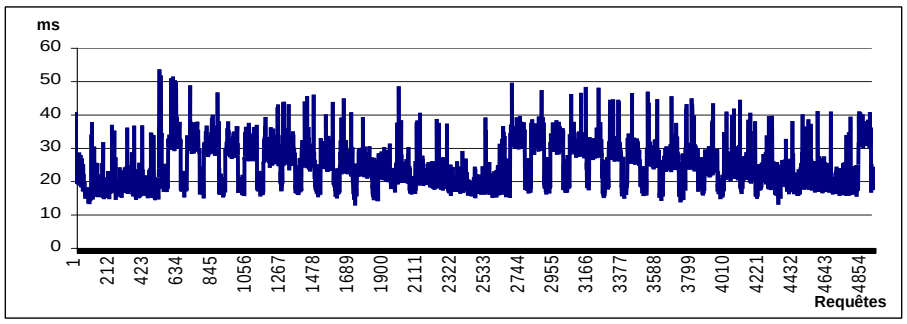


Figure 5.29. Temps de réponse de la découverte de contenu - 4 requêtes par seconde

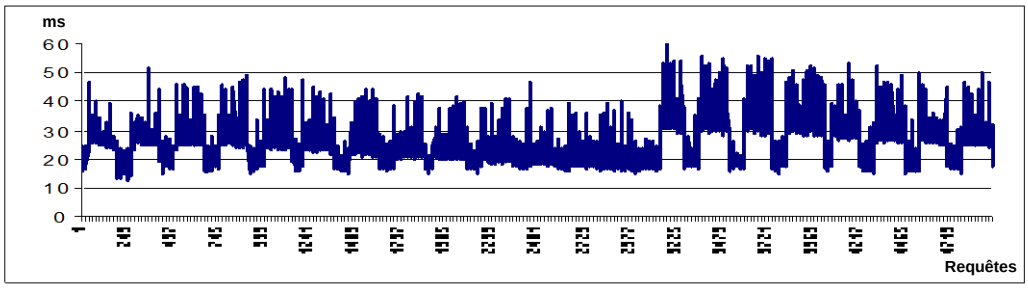


Figure 5.30. Temps de réponse de la découverte de contenu - 8 requêtes par seconde

Nous observons un comportement périodique qui se répète chaque 8mn. Dans chaque période, nous observons également un comportement similaire à celui repéré lors de la configuration locale. Le temps de réponse atteints des valeurs hautes pendant 20s ensuite des valeurs basses pendant 10s.

Nous remarquons que le temps moyen maximum est de 50ms alors que le temps moyen minimum est de 16ms. En effet, la configuration réseau augmente la valeur du temps de réponse.

Nous sommes surpris du nombre de réponses reçues par un nœud. Pour chaque requête, la découverte retourne jusqu'à 6 réponses. Le temps que nous conservons concerne seulement celui de la première réponse.

Discussion

Dans cette section, nous allons analyser les comportements observés et proposer des justifications.

Comportement périodique chaque 30s

Dans notre approche d'évaluation, nous garantissons que chaque nœud publie localement 100 annonces différentes dans son cache. Donc, si seul le nœud possédant l'annonce répond à une requête, il est logique que le nombre de réponses reçu par le nœud ayant émis la requête soit strictement égal à un. De plus la documentation de JXTA mentionne que si un nœud possède plusieurs annonces qui répondent à une requête, il ne retourne qu'une seule réponse contenant les annonces recherchées. Or, ce n'est pas le cas réellement car plusieurs réponses sont reçues (jusqu'à 6 réponses). Par conséquent, nous pouvons affirmer que plusieurs nœuds participent au résultat de la découverte. La question qui se pose est comment ces nœuds ont pu obtenir ces autres annonces ?

En lisant la documentation de l'interface *DiscoveryService* de l'API JXTA, nous avons trouvé que le service de découverte fourni par JXTA publie automatiquement et périodiquement les annonces trouvées dans le cache vers le groupe de pairs chaque 30s. Cette valeur correspond à la période de temps observée lors des expérimentations. La publication et la réception des annonces de contenu requièrent des ressources systèmes. C'est pourquoi, nous observons des temps de réponse élevés pendant 20s puis bas pendant 10s.

Pour vérifier ces propos, nous avons mesuré le temps nécessaire pour publier un répertoire contenant 100 objets de type *Content*. La publication dure environ 20s.

Une autre méthode consiste à récupérer le temps d'expiration inclus dans une annonce de contenu après la phase de découverte. Nous avons trouvé deux heures. La documentation JXTA nous confirme que le temps d'expiration des annonces est de un an pour un objet publié localement et de deux heures pour un objet publié à distance sur le réseau. Ainsi, l'annonce de contenu retrouvée après la phase de découverte ne provient pas du nœud propriétaire du contenu mais plutôt d'un autre nœud du réseau. Ce dernier a pu obtenir l'annonce grâce au processus de dissémination automatique et périodique fourni par le service de découverte de JXTA.

Comportement périodique chaque 8mn

Nous avons également observé un comportement périodique chaque 8mn. Ce comportement est plus clair au niveau de la configuration locale qui implique deux nœuds. Chacun émet périodiquement des requêtes de découverte et répond aux requêtes qui lui

parviennent. Au début de la période, le temps de réponse atteint des valeurs hautes. Cela signifie que ce nœud consomme une charge importante de ressources CPU et mémoire. Nous pensons donc que ce nœud traite plusieurs requêtes à la fois. Ensuite, le temps de réponse diminue jusqu'à la fin de la période. Cela signifie que le nœud consomme moins de ressources systèmes. Nous pensons que le nombre de requêtes à traiter à la fin de la période est très bas. Nous pouvons avancer l'hypothèse suivante :

Malgré qu'un nœud envoie périodiquement des requêtes de découverte, l'autre nœud les reçoit regroupées par ensemble de requêtes. Le traitement de chaque ensemble dure 8mn.

Cependant, cette hypothèse s'avère incorrecte, car le temps de réponse reste constant pendant la période où aucune dissémination n'a lieu (intervalle de 10s). Ceci signifie que les requêtes parviennent à un nœud de façon uniforme (à des intervalles constants) et non par ensemble. Si l'hypothèse était correcte, alors le temps de réponse aurait augmenté linéairement pendant l'intervalle de 10s. Ainsi, nous pensons que le service de découverte de JXTA retourne les résultats de découverte avec un temps de réponse constant. Concernant cette période de 8mn, nous pensons finalement qu'il s'agit d'une autre tâche qui requiert des ressources CPU et mémoire importantes.

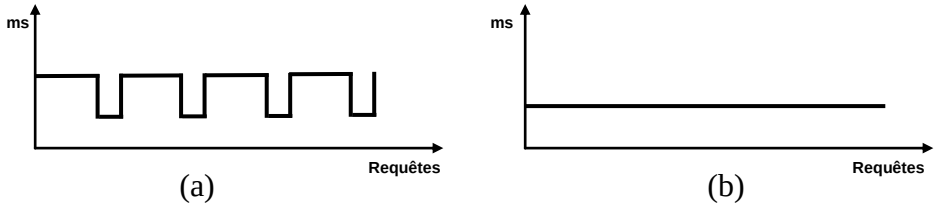


Figure 5.31. Temps de réponse

La figure 5.31 (a) montre une courbe où la tâche périodique de 8 mn n'est pas active. La figure 5.31 (b) montre une courbe où le processus de dissémination de JXTA n'est pas actif.

Configuration locale vs configuration LAN

Dans la configuration locale, seulement deux nœuds s'exécutant sur la même machine, participent aux expérimentations. Chacun émet vers l'autre des requêtes de découverte de façon périodique. Ainsi, chacun reçoit les requêtes selon des intervalles réguliers. C'est pourquoi les courbes sont cohérentes. Par contre, dans la configuration LAN, à cause de l'indéterminisme introduit par le réseau, un nœud reçoit les requêtes de découverte selon des intervalles irréguliers. Il peut traiter plusieurs requêtes à la fois puis seulement quelques requêtes. Ainsi, les temps de réponses sont variables comme le montrent les figures …

Enfin, comme attendu, les communications à travers le réseau augmentent les temps de réponse obtenus. Mais cette augmentation est minime (quelques millisecondes) et ne perturbe pas trop les performances du système.

5.4.2 Evaluation des capacités de rapatriement des contenus

Détails et résultats des expérimentations

L'objectif est d'évaluer les performances de rapatriement des contenus offerts par JXTA. Les expérimentations impliquent deux nœuds : un nœud fournisseur de contenus et un autre demandeur de contenus. Ces deux nœuds s'exécutent sur la même machine ayant 2.39 GHz

comme fréquence de microprocesseur et 504 Mo de mémoire vive. Cette évaluation est réalisée en local pour ne caractériser que les performances de la couche JXTA, sans ajouter de perturbation réseau.

Le nœud demandeur de contenus utilise l'API exportée par le module *demandeur proactif* pour envoyer périodiquement (chaque 1000ms) une requête de téléchargement d'un contenu directement vers le nœud fournisseur. Ensuite, il attend jusqu'à la réception du contenu demandé avant d'envoyer la requête suivante. La requête contient un identificateur de contenu.

Le nœud fournisseur de contenus utilise l'API exportée par le module *fournisseur réactif* pour attendre continuellement les requêtes de téléchargement. Pour chaque requête reçue, le nœud parcourt son dépositaire local et retourne une réponse contenant le contenu recherché vers le nœud demandeur. Ce dernier enregistre le temps écoulé entre l'émission de la requête et la réception d'une réponse.

Nous avons répété les tests plusieurs fois en faisant varier la taille des objets échangés : 16ko, 32ko, 64ko et 118ko. La figure suivante reporte le temps de réponse moyen.

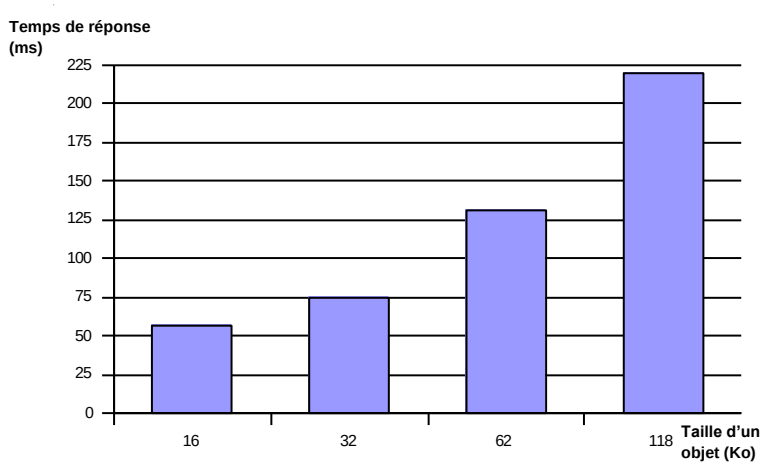


Figure 5.32. Mesure de performance du rapatriement de contenu

Discussion

En termes de fonctionnalité, l'étape d'acheminement des contenus permet effectivement de transférer un contenu depuis un nœud fournisseur vers un nœud demandeur. Les données recueillies indiquent des performances excellentes puisque les délais sont minimes (dans tous les cas inférieurs à 225ms) et les requêtes ont été résolues avec succès. Ceci souligne l'efficacité et la fiabilité du système réalisé.

Les résultats montrent que le temps de réponse est proportionnel à la taille des objets rapatriés. Ce comportement est normal puisque la préparation de l'émission des contenus dépend du nombre d'octets formant le contenu.

5.5 Conclusion

Ce chapitre a présenté l'ensemble des études et des réalisations que nous avons faites pour évaluer les performances d'une part de l'algorithme de déploiement et d'autre part de la plate-forme qui supporte cet algorithme. Nous avons d'abord décrit les différents prototypes que nous avons développés comme support à notre approche de déploiement Ensuite, nous avons rapporté les résultats concernant les phases de vérification de la compatibilité, de vérification de l'interopérabilité, de génération des configurations de déploiement et de

validation. Ces résultats nous ont permis de fixer de façon expérimentale les limites de notre algorithme. Ils ont prouvé la rapidité des traitements pour vérifier si une application satisfait ou non les contraintes de compatibilité et d'interopérabilité Ces étapes amont supportent jusqu'à 200 sites voisins et jusqu'à 600 implantations. Les résultats ont révélé aussi que les phases suivantes de génération et de validation des configurations de déploiement ne peuvent pas traiter plus que 19 implantations d'applications. Cependant, elles ne limitent pas aussi fortement l'algorithme car elles interviennent uniquement sur les implantations qui sont passée au travers des filtres formés par les deux étapes précédentes.

La deuxième partie des tests se focalise sur les performances de la plate-forme, en particulier sur les phases de découverte de contenu et de rapatriement. Les résultats obtenus ont montré un comportement périodique qui se répète chaque 8mn. De plus, dans chaque période, on retrouve un autre comportement périodique qui se répète chaque 30s. Ce dernier est dû au processus de dissémination automatique et périodique des annonces locales par le service de découverte de JXTA. Nous pensons que le premier comportement périodique est causé par une tâche répétitive qui nécessite beaucoup de ressources systèmes pour son exécution. Finalement, les résultats ont prouvé l'efficacité et la rapidité des traitements implantés.

Chapitre 6 Conclusion et perspectives

6.1 Déploiement et reconfiguration dans les sessions collaboratives

Dans cette thèse, nous avons travaillé sur la problématique du déploiement et de la reconfiguration des applications dans les sessions collaboratives synchrones. Ce sujet est fortement d'actualité aujourd'hui, surtout avec les avancées réalisées tant au niveau de l'infrastructure réseaux, qu'au niveau des terminaux de déploiement (ordinateurs de bureau, portables, PDAs,...). La majorité des travaux autour du déploiement se focalisent soit sur le déploiement des applications sur un seul nœud indépendamment de son contexte global, soit sur le déploiement des applications distribuées orientées composant dans le cas de plusieurs nœuds interconnectés. Rares sont les travaux traitant le déploiement des applications en prenant en compte le contexte local ainsi que le contexte global. De plus, du point de vue architectural, ces travaux se basent essentiellement sur un contrôle centralisé limitant ainsi les opportunités du travail collaboratif.

Dans cette thèse, nous avons présenté notre approche pour le déploiement et la reconfiguration dans le cas des sessions collaboratives. Cette approche se base sur trois éléments : (1) des modèles abstraits permettant d'exécuter automatiquement les opérations de déploiement et de reconfiguration. (2) un algorithme décentralisé capable de générer des configurations de déploiements couvrant les besoins d'un participant à la session. (3) Une plate-forme générique supportant l'algorithme proposé.

Le troisième chapitre a été dédié à la présentation des modèles abstraits et de l'algorithme de déploiement et de reconfigurations. Ces modèles décrivent la structure de la session, les applications, ainsi que le nœud de déploiement. Le modèle de session se base sur un graphe orienté. Les nœuds du graphe font référence aux participants de la session. Les étiquettes sur les arcs décrivent les flux de données échangés entre les participants. Le modèle d'application permet de classer les applications en trois catégories selon le flux de données qu'elles traitent. Ainsi, on retrouve dans la première catégorie, les applications qui sont constituées d'une seule entité monolithique et qui traitent un seul type de données en production ou en consommation. Dans la deuxième catégorie, on retrouve les applications composées de deux entités traitant ainsi un seul flux de données mais en production et en consommation à la fois. La troisième catégorie est plus générale. Les applications appartenant à cette catégorie sont formées de plusieurs entités permettant l'envoi et la réception des flux de données issus de différents types. Ce modèle permet d'exposer les exigences d'une application en termes des ressources requises pour son bon fonctionnement. Le modèle de nœud permet de décrire un nœud de déploiement grâce à une sémantique simple qui reflète les caractéristiques logicielles et matérielles du nœud.

L'algorithme proposé permet à un nœud de réaliser les opérations de déploiement et de reconfiguration de façon autonome tout en gardant des relations avec ces voisins. Ces relations lui seront utiles pour récupérer les entités décrivant les configurations déployées sur

les nœuds voisins et ainsi pouvoir choisir les applications respectant les critères d'interopérabilité. La particularité de cet algorithme est qu'il permet de générer des configurations de déploiement valides. Ces configurations décrivent les applications permettant de couvrir les besoins d'un participant tout en étant compatibles à l'environnement d'exécution et interopérables avec les applications déjà déployées sur les nœuds voisins.

Dans le chapitre quatre, nous avons présenté notre plate-forme supportant l'algorithme de déploiement et de reconfiguration. Cette plate-forme se base en une architecture en couches. La couche inférieure est constituée par JXTA qui offre un ensemble de primitives de base communes à toutes les applications Pair-à-Pair. La couche en dessus offre des modules permettant de réaliser les opérations de création de contenu générique, de publication des annonces dans un réseau Pair-à-Pair, de découverte et enfin de rapatriement des contenus depuis le nœud fournisseur vers le nœud demandeur. La couche suivante est orientée déploiement. Elle profite des modules de la couche inférieure pour permettre de réaliser les opérations de déploiement et de reconfiguration.

Enfin dans le cinquième chapitre, nous avons utilisé la plate-forme précédente pour développer des prototypes qui ont servi à valider notre algorithme. Nous avons ensuite mené une étude de performance pour évaluer d'une part les principales phases de déploiement et de reconfiguration, à savoir la vérification de la compatibilité, la vérification de l'interopérabilité, la génération des configurations de déploiement et la validation de ces configurations. D'autre part, les évaluations ont permis de mesurer les capacités de la plate-forme pour effectuer la découverte des contenus et leurs rapatriements. Les résultats ont montré que l'implantation actuelle de l'algorithme permet de vérifier la compatibilité et l'interopérabilité très rapidement. Il peut supporter jusqu'à 600 applications. Par contre, le temps nécessaire pour générer les configurations de déploiement et la validation à un comportement exponentiel et dépend fortement du nombre des applications. Ces phases interviennent à la suite des phases précédentes, et s'appliquent ainsi sur un nombre plus petit d'applications précédemment filtrées, nombre inférieur à 19.

Concernant les opérations de découverte de contenu, les tests ont révélé un comportement périodique qui se répète chaque 8mn. Dans chaque période, nous avons observé un autre comportement périodique qui se répète chaque 30s. Le temps de réponse atteint des valeurs hautes pendant 20s puis des valeurs basses pendant 10s. Nous avons vérifié que ce dernier comportement est dû au processus de dissémination automatique et périodique des annonces locales. Le premier comportement périodique est probablement lié à une tâche périodique indépendante du service de découverte de JXTA. Les dernières expérimentations se focalisent sur les capacités de rapatriement de contenu entre les nœuds du réseau Pair-à-Pair. Comme attendu, le temps de transfert dépend de la taille des contenus. Cependant, les résultats ont montré des temps acceptables et encourageants.

6.2 Perspectives

Le travail que nous avons réalisé dans cette thèse a permis, d'un côté de fournir une analyse relative à la problématique du déploiement et de la reconfiguration, et d'un autre côté de proposer une solution pour déployer les applications requises dans les sessions collaboratives. Ce travail peut être amélioré selon trois directions :

6.2.1 Modèle de session

La première perspective de recherche concerne le modèle de session. Dans notre approche, la localisation des applications se base sur le couple {type de données, sens de communication}. Ces informations sont obtenues à partir des arcs qui relient les nœuds du graphe de session. Ces critères de recherche peuvent retourner des applications appartenant à des domaines très variés. Considérons le scénario suivant, dans lequel une session est formée par trois participants :

L'instance du modèle de session est représentée par la figure 6.1.

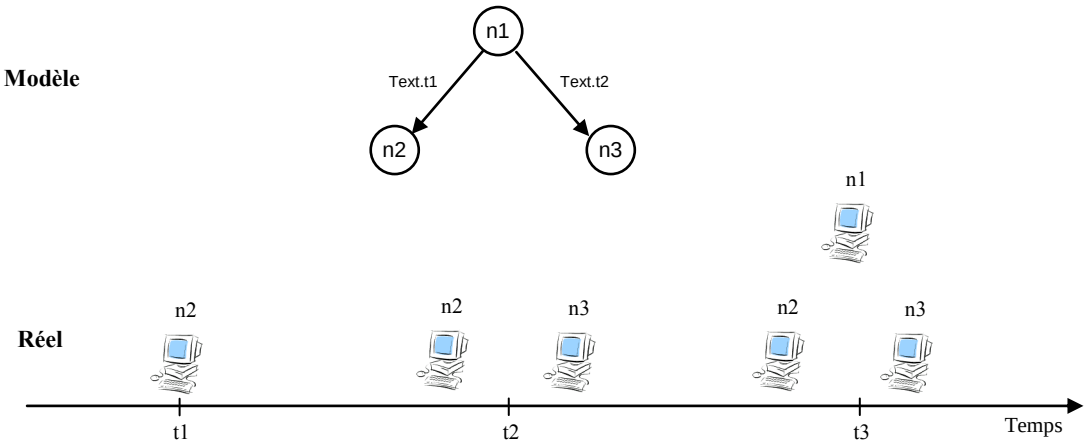


Figure 6.1. Scénario de connexion

Les applications disponibles sont :

- Dialogue permettant de produire et de consommer des flux textuels au format ASCII,
- Navigateur qui peut envoyer et recevoir des flux textuels au format ASCII,
- Messagerie. Les messages échangés sont au format ASCII.

- A l'instance t1, le nœud n2 se connecte à la session. En utilisant le graphe, l'algorithme de déploiement tente de trouver les applications permettant de produire un flux textuel. Comme il n'y a aucun participant déjà connecté, alors l'algorithme choisit par exemple l'application Messagerie.

- A l'instant t2, le nœud n3 se connecte. De la même manière, l'algorithme peut choisir l'application Navigateur.

- A l'instant t3, le nœud n1 se connecte à la session. Les applications à déployer doivent être interopérables avec celles installées chez n2 et n3. Puisque le format des flux de données est le format ASCII, alors l'algorithme de déploiement peut choisir parmi les trois applications disponibles. Par exemple, il peut choir l'application Dialogue. On se retrouve donc avec la configuration suivante :

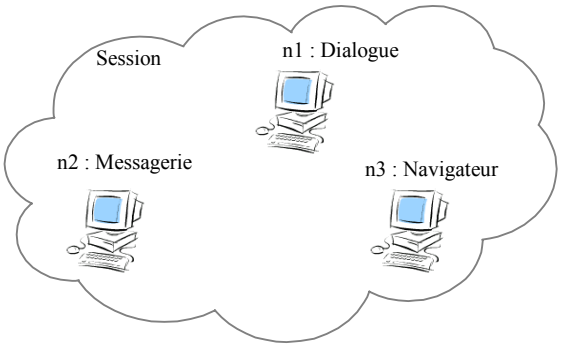


Figure 6.2. Configuration réelle de la session

Avec uniquement ces informations, la session est cohérente par rapport à l'algorithme de déploiement. Cependant, les participants ne peuvent pas communiquer entre eux car les applications appartiennent à des domaines différents. Ce problème peut se résoudre facilement en étendant la sémantique associée aux arcs reliant les nœuds du graphe de session. Par exemple, si le fait de qualifier le flux textuel d'interactif élimine l'outil messagerie. De plus, lors de la description des applications, il est nécessaire de leur associer des listes de protocoles supportés pour les flux de données pour mieux caractériser leur interopérabilité. On recherche alors les applications par leurs propriétés et leurs caractéristiques, puis on vérifie leur interopérabilité de manière plus fine. Ces extensions sont facilement implantables en ajoutant des attributs aux graphes de session proposés, attributs réalisés sous la forme de balises XML.

6.2.2 Variation de l'environnement d'exécution

Un axe de recherche que nous n'avons pas abordé jusqu'à maintenant et qui fait partie des extensions possibles de nos travaux concerne la sensibilité active. La version actuelle de l'algorithme permet de traiter seulement l'évolution de la session c'est-à-dire l'entrée et la sortie d'un utilisateur ainsi que le changement de rôle. Il serait judicieux de proposer une solution permettant aussi de prendre en compte l'évolution du contexte local c'est-à-dire les variations des ressources offertes par le nœud de déploiement. Cet axe de recherche fait partie des thèmes de recherche récents et émergents liés à l'auto-adaptation [48], [49].

Dans le cas où l'algorithme prend en compte l'évolution du contexte local alors la configuration de déploiement sur chaque nœud voisin est susceptible d'évoluer. Les applications installées risquent de ne plus être interopérables avec celles nouvellement déployées sur les nœuds voisins. Il est donc préférable que l'algorithme tienne compte aussi de l'évolution des contextes locaux des nœuds voisins pour qu'il permette de maintenir l'interopérabilité entre les différentes applications.

6.2.3 Etude de performance

Les expérimentations actuelles relatives à la couche noyau générique, ont été réalisées sur un nombre assez limité de nœuds. La technologie Pair-à-Pair suppose que plus de nœuds peuvent intervenir. Il serait donc intéressant d'évaluer les services de découverte de contenus et de rapatriement dans le cas de groupe de pairs regroupant un nombre très grand de participants. De plus le modèle Pair-à-Pair supporte des topologies plus riches que le client-serveur [50]. Nous trouvons essentiellement trois topologies :

- Topologie P2P avec serveur central. Les nœuds se connectent à un serveur central lors du démarrage et publient les contenus qu'ils désirent partager. Lors de la découverte, un nœud envoie une requête au serveur. Ce dernier consulte son index et retourne une liste des annonces des contenus respectant la requête.

- Topologie P2P totalement décentralisée. Les nœuds ont un comportement identique. Diverses techniques de routage ont été proposées pour faciliter la découverte mais la plupart se basent sur la propagation de la requête. Ainsi, un nœud qui reçoit la requête, consulte son cache avant de la diffuser vers les autres nœuds. Dans le cas où il dispose des contenus recherchés, il retourne une liste d'annonces vers le nœud demandeur.

- Topologie P2P basée sur des supers nœuds. Dans cette topologie, un ensemble de nœuds puissants et assez stables, forme une couche Pair-à-Pair dans laquelle la découverte se fait par dissémination. Chaque super nœud réunit un ensemble de nœuds. Ainsi, un nœud envoie une requête vers son super nœud. Ce dernier consulte son cache et diffuse la requête vers les autres supers nœuds.

Certes, les temps de réponse lors de la phase de découverte de contenus varient selon la topologie adoptée. Définir et mettre en place un ensemble de moyens matériels et logiciels pour évaluer les performances de la découverte est une perspective ambitieuse et très prometteuse.

6.2.4 Implantation réelle de la reconfiguration

Actuellement, nous avons proposé une solution théorique pour la reconfiguration lors de l'évolution des rôles des utilisateurs (section 3.4.3). Ainsi, deux phases composent la reconfiguration. Dans la première phase, l'algorithme s'exécutant sur chaque nœud, détermine les nouveaux besoins de ce nœud ainsi que le sous-ensemble des nœuds impliqués dans l'évolution. La deuxième phase se base sur un mécanisme de jeton permettant de n'avoir qu'un seul nœud à un instant donné, en train de faire les opérations du déploiement ultérieur. L'implantation de cette solution a été réalisée seulement sur le simulateur de déploiement. Les résultats sont prometteurs.

L'implantation réelle de cette solution requiert des mécanismes permettant de coordonner les opérations entre les nœuds. En particulier, il serait intéressant de définir une API pour mettre en pratique les fonctionnalités de passage de jeton.

Publications

Publications principales

AINA 2007 — E. Hammami, T. Villemur, “Design of a decentralized algorithm for deploying services within netwtorked nodes”, The 21st International Conference on Advanced Information Networking and Applications (AINA’07), Niagara Falls, Canada, May 21-23, 2007.

ECUMN 2007 — E. Hammami, “Towards a Peer-to-Peer Content Discovery and Delivery Architecture for Service Provisioning”, 4th European Conference on Universal Multiservice Networks (Ecumn’2007), February 14-16, 2007, Toulouse, France.

AT 2006 — E. Hammami, T. Villemur, “Plate-forme pour le déploiement coopératif sensible au contexte”, Journal des Annales des Télécommunications, Vol 61/11-12 – 2006.

NOTERE 2005 — E. Hammami, T. Villemur, “Déploiement adaptatif des composants dans les sessions collaboratives”, NOuvelles TEchnologies de la Répartition (NOTERE’2005), August, 29 to 1st of September 2005, Gatineau, Canada.

ISSADS 2005 — E. Hammami, T. Villemur, K. Drira, “An online component deployment system for dynamic collaborative sessions”, Fifth IEEE International Symposium and School on Advance Distributed Systems (ISSADS’2005), Guadalajara, Jalisco, México, January 24-28, 2005.

Autres publications

JDIR 2005 — E. Hammami, T. Villemur, “Nouvel algorithme pour le déploiement sensible au contexte”, 7èmes Journées Doctorale Informatique et Réseau (JDIR'2005), Troyes, France, 14-15 December, 2005, pp.47-64.

GEI 2005 — E. Hammami, T. Villemur, K. Drira, ”Service pour la Génération de Configurations de Déploiement”, 5èmes Journées Scientifiques des Jeunes Chercheurs en Génie Electrique et Informatique (GEI'2005), Sousse, Tunisie, March 25-27, 2005.

WCC 2004 — E. Hammami, T. Villemur, K. Drira, “Design of a component deployment service for dynamic collaborative sessions”, IFIP 18th World Computer Congress (WCC’2004). Student Forum, Toulouse, France, August 2004, pp.205-216.

GEI 2004 — E. Hammami, T. Villemur, K. Drira, “Service de déploiement dynamique de composants coopératifs dans un environnement pair-à-pair”, 4èmes Journées Scientifiques des Jeunes Chercheurs en Génie Electrique et Informatique (GEI'2004), Monastir, Tunisie, March 15-17, 2004.

Bibliographie

[1] RODRIGUEZ PERALTA L.M., VILLEMUR T., DRIRA K., *et al*.
Managing dependencies in dynamic collaborations using coordination diagrams
Proceedings of 6th International Conference on Principles of Distributed Systems (OPODIS'02), pp.29-42, Reims (France), 11 – 13 Décembre, 2002.

[2] ARSHAD N., HEIMBIGNER D., WOLF L. A.
Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems.
Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2003.

[3] AYED D., TACONET C., SABRI N., *et al*.
Plate-forme de déploiement sensible au contexte des applications à base de composants.
4ème Conférence Française sur les Systèmes d'Exploitation CFSE'4, Le Croisic France, 6-8 Avril 2005.

[4] HALL R. S., HEIMBIGNER D., WOLF A. L.
A Cooperative Approach to Support Software Deployment Using the Software Dock.
International Conference on Software Engineering ICSE'99, Los Angeles, CA, Mai 1999, pp. 174-183.

[5] LESTIDEAU V., BELKHATIR N.
Providing Highly automated and generic means for software Process deployment.
In the proceedings of 9th European Workshop on Software Process Technology EWSPT 2003, Helsinki, Finland, Septembre 2003.

[6] SATOH I.
Dynamic deployment of pervasive services
Proceedings of the International Conference on Pervasive Services (ICPS'05), p 302-311, 11-14 July 2005.

[7] FRENOT S.
Gestion du déploiement de composants sur réseau P2P.
1ère Conférence Francophone sur le Déploiement et la (Re) Configuration de Logiciels Décor'2004, Grenoble, 28-29 octobre 2004.

[8] ROUSSAIN H., GUIDEC F.
Cooperative Component-Based Software Deployment in Wireless Ad Hoc Networks
In 3rd International Working Conference on Component Deployment CD 2005, LNCS, Grenoble, France, Novembre 2005.

[9] KETFI A., BELKHATIR N., CUNIN P.Y.
Adaptation Dynamique Concepts et Expérimentations
Proceedings of the 15th International Conference on Software & Systems Engineering and their Applications ICSSEA'02, Paris, France, December 2002 (8 pages).

[10] KEPHART J.O., CHESS D.M.
The vision of autonomic computing
IEEE Computer, Vol. 36, No 1, pp 41-50, Janvier 2003.

[11] PARASHAR M., HARIR S.
Autonomic computing : an overview
Unconventional Programming Paradigms, Lecture Notes in Computer Science (LNCS), Vol 3566/2005, pp. 257-269, April 2005.

[12] DEY A.K., ABOWD G.D.
Towards a Better Understanding of Context and Context-Awareness
In the Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the *2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, April 3, 2000. Also GVU Technical Report GIT-GVU-99-22. Submitted to the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99), June 1999.

[13] DEY A.K., ABOWD G.D., SALBER D.
A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.
Human-computer interaction, Vol. 16, No 2, 3 & 4, pp. 97-166, 2001.

[14] LI L., HORROCKS I.
A software framework for matchmaking based on semantic web technology.
In Proceedings of the 12th International World Wide Web Conference (2003), p. 331-339

[15] Java Web Start
http://java.sun.com/products/javawebstart, 2006.

[16] InstallShield
http://www.installshield.com, 2006.

[17] Microsoft Windows Installer
http://www.microsoft.com, 2006.

[18] EWING M., TROAN E.
The RPM Packaging System.
In Proceedings of the First Conference on Freely Redistributable Software, Cambridge, MA, USA, February 1996. Free Software Foundation.

[19] Hewlett-Packard Company, HP-UX Release 10.10 Manual, November 95.

[20] CARZANIGA A., FUGGETTA A., HALL R.S., *et al*.
A Characterization Framework for Software Deployment Technologies.
Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.

[21] FRENOT S., ROYON Y.
Component deployment using a Peer-to-Peer overlay.
In Proc. Of 3rd International Working Conference on Component Deployment (CD'05), Lecture Notes in Computer Science (LNCS), Vol. 3798/2005, pp. 185-198, Springer, Novembre 2005.

[22] VANISH T., DEJAN M., QINYI W., *et al*.
Approaches for Service Deployment,
IEEE Internet Computing, Service-Oriented Computing Track, Mars-Avril 2005.

[23] Nixes.
www.aqualab.cs.northwestern.edu/nixes.html, 2006.

[24] SmartFrog.
http://www.smartfrog.org, 2006.

[25] Radia.
http://www.novadigm.com, 2006.

[26] FLISSI A., MERLE P., GRANSART C.
A Component-based Software Infrastructure for Ubiquitous Computing.
The 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005), Lille, France, 4-6 July 2005.

[27] AYED D., TACONET C., BERNARD G.
Deployment and Reconfiguration of Component-based Applications in AMPROS.
Proactive computing workshop (PROW 2004) - Helsinki, Finland - November 25-26, 2004.

[28] AYED D., TACONET C., BERNARD G., BERBERS Y.
An adaptation methodology for the deployment of mobile component-based applications.
IEEE International Conference on Pervasive Services 2006 (ICPS 2006), Lyon, France, 26-29 Juin 2006, pp. 193-202.

[29] FLISSI A., MERLE P.
A generic deployment framework for grid computing and distributed applications.
In Proc. Of International Conference on Grid Computing, High Performance and Distributed Applications (GADA'06), Lecture Notes in Computer Science (LNCS), Vol. 4276/2006, pp. 1402-1411, Novembre 2006.

[30] BELOUED A., TACONET C., AYED D., *et al*.
Placement automatique des composants lors du déploiement d'applications à base de composants.
Journée Composants (JC 05) - Le Croisic France - 6-8 Avril, 2005.

[31] SEVILLA D., GARCIA J.M., GOMEZ A.
Design and Implementation Requirements for CORBA Lightweight Components.
2001 International Conference on Parallel Processing Workshops (ICPPW'01), p. 0213.

[32] BELOUED A.
Applications multi-composants et déploiement sur terminaux mobiles.
Rapport de stage. DEA Méthodes Informatiques des Systèmes Industriels. Septembre 2004.

[33] Resource Description Framework.
http://fr.wikipedia.org/wiki/RDF

[34] Ontology Web Language.
http://fr.wikipedia.org/wiki/Web_Ontology_Language

[35] COAH, Specification of the Deployment and Configuration.
http://coach.objectweb.org

[36] Microsoft Office Groove 2007
http://office.microsoft.com/fr-fr/groove/default.aspx

[37] BOOCH G., RUMBAUGH J, JACOBSON I.
The Unified Modeling Language User Guide, 2nd edition, Addison-Wesley Professional, pp. 496, 2005.

[38] Wikipedia Free Encyclopedia. Extensible Markup Language (XML)
http://en.wikipedia.org/wiki/XML

[39] VILLEMUR T.
Conception de services et de protocoles pour la gestion de groupes coopératifs. 166p.
Th. : Informatique : Toulouse : 1995 ; 1952.

[40] RODRIGUEZ PERALTA L.M., VILLEMUR T., DRIRA K.
An XML on-line session model based on graphs for synchronous cooperative groups.
Proceedings of the international Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'2001, Volume III, Session: Coordination in Component-Oriented Systems Architecture, Models, Languages and Applications, Las Vegas, Nevada, USA, pages 1257-1263, June 25 - 28, 2001.

[41] RODRIGUEZ PERALTA, Laura Margarita
Service de gestion de session orienté-modèle pour des groupes collaboratifs synchrones. 166p.
Th : Informatique : Toulouse : 2003 ;

[42] HAMMAMI E., VILLEMUR T.
Design of a decentralized algorithm for deploying services within networked nodes.
The 21st International Conference on Advanced Information Networking and Applications (AINA'07), Niagara Falls, Canada, May 21-23, 2007.

[43] HAMMAMI E., VILLEMUR T.
Plate-forme pour le déploiement coopératif sensible au contexte.
Journal des Annales des Télécommunications, Vol 61/11-12 – 2006.

[44] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY, 1990.

[45] GONG L.
JXTA: A Network Programming Environment.
IEEE Internet Computing, May 2001, p. 88-95.

[46] EHRIG M., SCHMITZ C., STAAB S., *et al*.
Towards Evaluation of Peer-to-Peer-based Distributed Information Management Systems.
AAAI Spring Symposium on Agent-Mediated Knowledge Management(AMKM – 03), March 24-26, 2003

[47] WALKERDINE J., MELVILLE L., SOMMERVILLE I.
Dependability within Peer-to-Peer Systems.
In the proceedings of the ICSE Workshop on Architecting Dependable Systems (WADS) 2004, Edinburgh, 25 May, 2004.

[48] GARLAN D., CHENG S.-W, HUANG A.-C., *et al*.
Rainbow: architecture-based self-adaptation with reusable infrastructure.
IEEE Computer, Vol. 37.

[49] SATOH I.
Self-organizing software components in Distributed Systems.
In Proc. of 20th International Conference on Architecture of Computing Systems. Aspects in Pervasive and Organic Computing (ARCS'07), Lecture Notes in Computer Science (LNCS), Vol. 4415/2007, pp. 185-198, Springer, Mars 2007.

[50] SINGH M.P.
Peering at Peer-to-Peer Computing
IEEE Internet Computing, Vol. 5, N° 1, 2001, pp. 4-5.

Context-aware deployment and reconfiguration of applications within collaborative sessions

Abstract

The processes of context-aware deployment and reconfiguration of applications within collaborative sessions manage initial and later distribution of collaborative tools within participant nodes while taking into account various constraints. These constraints come from the session structure, the running environments and the relationships among participant nodes.

The majority of works related to deployment and reconfiguration topics propose solutions that mainly address the two first points without giving enough importance to the last point. Thus, the deployed applications satisfy user's needs and are compatible with local context but interoperability with already deployed applications on neighbouring nodes is not verified. Moreover, the control of deployment and the discovery of required applications are made in a centralized way and require human intervention.

We propose an approach for automatic deployment and reconfiguration based on a decentralized algorithm that runs on each deployment node. This algorithm uses abstract models to generate valid deployment configurations respecting all constraints. A Peer-to-Peer platform has been developed. It is composed of generic modules to implement this algorithm. Finally, several prototypes have been implemented. Performance assessments have been led in order to evaluate our approach.

Keywords

Context-aware deployment, reconfiguration, collaborative session, Peer-to-Peer, JXTA