Architecture pour les systèmes de déploiement logiciel à grande échelle : prise en compte des concepts d'entreprise et de stratégie

Noelle Merle

UNIVERSITE JOSEPH FOURIER

THESE

pour obtenir le grade de

DOCTEUR de l'Université Joseph Fourier de Grenoble

Discipline : Informatique

présentée et soutenue publiquement par

Noëlle MERLE

Le 1er décembre 2005

ARCHITECTURE POUR LES SYSTEMES DE DEPLOIEMENT LOGICIEL A GRANDE ECHELLE : PRISE EN COMPTE DES CONCEPTS D'ENTREPRISE ET DE STRATEGIE

JURY

Y. Chiaramella	Professeur Université J. Fourier, Grenoble	(Président)
D. Hagimont	Professeur ENSEEIHT, INP Toulouse	(Rapporteur)
M. Oussalah	Professeur Université de Nantes	(Rapporteur)
J.-M. Andreoli	Directeur de recherche Xerox Research Centre Europe	(Examinateur)
P.-Y. Cunin	Professeur Université J. Fourier, Grenoble	(Directeur de thèse)
N. Belkhatir	Professeur Université P.-M. France, Grenoble	(Directeur de thèse)

à Claire,

A défaut d'avoir fait un long discours le jour de ma soutenance, pour ne pas me laisser submerger par les émotions, je vais tenter de faire ici quelques remerciements plus personnels et plus chaleureux...

Je vais commencer par remercier chacun des membres du jury, Yves Chiaramella, Daniel Hagimont, Mourad Chabane Oussalah et Jean-Marc Andreoli, pour m'avoir accorder de leur temps si précieux et m'avoir fait profiter de leur expérience. Merci à vous d'avoir évalué ce travail avec autant de justesse et de pertinence.

Je remercie aussi mes directeurs de thèse, Pierre-Yves Cunin et Noureddine Belkhatir, d'avoir été présents tout au long de ces années, depuis mon DEA à la rentrée 2001. Merci à toi Pierre-Yves pour ton soutien dans mes travaux, pour ces discussions et ces remarques toujours pertinentes et constructives qui font avancer. Merci à toi Nourry pour m'avoir encadrée.

Je vais continuer en remerciant chacun des membres de l'équipe Adèle, ceux qui sont déjà partis vers d'autres horizons comme ceux qui sont encore là, pour cette bonne ambiance qui règne au sein de l'équipe. Un remerciement plus particulier pour ceux qui ont participé, de près ou de loin, à quelques discussions « philosophiques » pour faire progresser ce travail : Anca, German, Jorge, Vincent, ... Merci aussi à Madjid, mon « collègue de bureau », pour ces discussions passées à refaire le monde et cet enrichissement culturel quasi-quotidien.

Je remercie aussi Philippe Martin et Camille Belissant, du département informatique de l'IUT2, pour m'avoir fait confiance et donner ma chance dans l'enseignement. Cette expérience fut très enrichissante pour moi.

Merci aussi à tous ceux qui ont relu ce document dans son intégralité ou en partie, que ce soit pour le « fond » ou la « forme », afin de l'amener à ce stade. J'ai les mêmes pensées pour ceux qui m'ont fait des remarques pour améliorer la présentation de ma soutenance.

Et puis, une thèse, c'est des bons moments, mais aussi des moins bons... Et dans les moments de doute comme dans les moments de joie, j'ai été heureuse de pouvoir compter sur quelques « piliers »... Je pense bien sûr à ma famille et mes ami(e)s.

Merci à Damien pour tellement de choses, depuis les découvertes culinaires jusqu'aux plaisanteries, depuis les découvertes montagnardes jusqu'au réconfort des moments de doute, en passant par tout le reste! Merci à Virginie pour ce sourire et cette bonne humeur qui déplaceraient des montagnes! Merci aussi à Christophe, Anne, Corinne, Christine, et quelques autres, d'avoir été là à un moment ou à un autre... tout simplement! J'ai aussi une pensée pour Céline qui n'a rien oublié, et surtout pas de prendre de mes nouvelles régulièrement.

Et les derniers, mais non les moindres, ce sont ceux qui sont là depuis toujours et qui m'ont soutenue dans tout ce que j'ai pu entreprendre, mes parents et mon frère Jean-Michel. C'était à la fois très touchant et un peu gênant de lire toutes ces choses dans vos regards... Ce travail, c'est aussi un peu de chacun de vous...

Et je ne pourrais pas terminer cette page sans avoir une pensée très émue pour ceux qui nous ont quittés beaucoup trop tôt, et en particulier, pour ma sœur Claire... Si je suis là aujourd'hui, c'est aussi pour et grâce à toi…

RESUME

Le déploiement arrive en fin de cycle de vie du logiciel. Cette étape a pris une grande importance dans les entreprise ayant un grand nombre de machines à gérer : on parle alors de déploiement à grande échelle. Les entreprises souhaitent pouvoir automatiser cette étape, pour éviter d'avoir à répéter les mêmes étapes plusieurs fois. Cependant, elles souhaitent garder un certain contrôle sur les activités de déploiement, en imposant différentes contraintes (même version d'une application sur un groupe de machines, préférence pour des applications avec certaines caractéristiques, ordonnancement prioritaire, …). Dans notre travail, nous définissons un méta-modèle de déploiement permettant d'automatiser le traitement des données liées au déploiement. Nous définissons aussi des stratégies de déploiement qui permettent aux entreprises de définir leur propres contraintes de déploiement, selon leurs besoins et leurs préférences.

MOTS-CLES

Déploiement à grande échelle, entreprise, stratégies de déploiement, méta-modèle de déploiement, ORYA.

ABSTRACT

Deployment comes at the end of software lifecycle. This step is more and more important in enterprises managing a lot of machines : we speak about large scale deployment. Enterprises wants to automate this step, in order to not repeat the same steps several times. However they wishes to keep control on deployment activities, forcing different constraints (same version of an application on a group of machines, preference for applications with fixed characteristics, specific scheduling, …). In our work, we define a deployment metamodel, allowing to automate the processing of deployment data. We also define deployment strategies which allow enterprises to define their own deployment constraints, according to their needs and preferences.

KEYWORDS

Large scale deployment, enterprise, deployment strategies, deployment metamodel, ORYA.

TABLE DES MATIERES

TABLES DES FIGURES

CHAPITRE 1 : INTRODUCTION

Préambule :

Ce chapitre présente l'idée générale de notre travail : le contexte, les objectifs et la méthodologie.

---

1. CONTEXTE DE TRAVAIL

Le déploiement est l'étape qui arrive en « fin » du cycle de vie du logiciel. Il s'agit d'une étape complexe qui permet non seulement d'installer le logiciel, mais aussi de le gérer jusqu'à la fin de sa vie. Plusieurs activités entrent alors en jeu, comme la mise à jour ou la désinstallation. Il s'agit alors de prendre en charge le logiciel depuis son installation, jusqu'à sa désinstallation, en passant par toutes les phases de maintenance (évolutive ou corrective) qui séparent ces deux activités.

Cette étape a pris de l'importance ces dernières années. En effet, avec le développement d'Internet, il est maintenant possible de déployer un produit directement depuis son producteur vers une machine cible définie.

A une échelle unitaire, le déploiement est une étape qui reste gérable. En effet, un utilisateur qui connaît bien les caractéristiques de sa propre machine est capable de choisir les produits qu'il souhaite déployer. Ainsi, par exemple, s'il a une machine fonctionnant sous *Windows*, il choisit une version d'un traitement de texte qui est adaptée et qui remplit cette contrainte. Il ne lui reste ensuite qu'à utiliser un CD ou un fichier d'installation qu'il a téléchargé depuis Internet. Le produit fonctionne ensuite correctement (aux erreurs de manipulation près).

Cependant, dès que le nombre de machines à gérer augmente, cette étape devient moins évidente. Elle devient d'autant plus complexe lorsque les machines ont des caractéristiques différentes. Les entreprises ont alors des difficultés à pouvoir gérer « manuellement » l'ensemble de leur parc informatique. En effet, l'entreprise peut, par exemple, posséder des machines utilisant des systèmes d'exploitation, des capacités mémoire ou des processeurs différents. Il faut alors choisir une version différente pour chaque machine.

Pour les entreprises ayant un nombre important de machines, il est aussi très contraignant de devoir se déplacer sur chaque machine et de répéter les mêmes étapes plusieurs fois, lorsque qu'un déploiement est nécessaire. Un administrateur ne peut pas laisser non plus l'entière charge du déploiement à chaque utilisateur. D'abord, certains utilisateurs utilisent leur matériel informatique comme un simple outil de travail et ne peuvent pas déterminer quelle version d'un produit peut ou non fonctionner correctement sur leur machine. Ensuite, il faut parfois gérer la compatibilité de versions d'un produit entre plusieurs machines. Par exemple, il est préférable d'imposer la même version d'un compilateur au groupe des développeurs de logiciels, ou la même version d'un traitement de texte à toute l'entreprise pour faciliter l'échange de documents.

Il est nécessaire de proposer aux entreprises un environnement qui sert de support pour le déploiement. Un tel environnement permet d'automatiser le déploiement. Cette automatisation permet non seulement d'éviter les étapes répétitives de chaque déploiement unitaire, mais aussi de limiter les problèmes de dysfonctionnement dus au déploiement de produits incompatibles avec la configuration des machines cibles.

Les entreprises peuvent aussi souhaiter personnaliser leurs activités de déploiement, pour assurer un certain degré de qualité à ces opérations. Par exemple, elles peuvent privilégier les produits implémentés en java, ou ceux fournissant une interface graphique. Elles peuvent imposer un ordre de déploiement : par exemple, les produits doivent être déployés en priorité sur les machines utilisées par les chefs de services de l'entreprise. Elles peuvent aussi

souhaiter assurer un certain niveau de sécurité, etc. On parle alors de stratégies de déploiement.

Les stratégies permettent de gérer la flexibilité liée au déploiement. En effet, plusieurs versions d'un produit peuvent être disponibles et compatibles avec une machine cible. Il s'agit alors d'en sélectionner une qui correspond au mieux aux préférences de l'utilisateur ou de l'entreprise (type d'interface, langage d'implémentation, langue de l'utilisateur, etc.). Lorsque le déploiement concerne un ensemble de machines, un ordonnancement particulier peut être choisi par l'administrateur. Différentes stratégies peuvent aussi être utilisées pour établir quoi faire en cas d'erreur au cours du déploiement (retour en arrière, activité de compensation, …).

Le déploiement est un domaine qui regroupe un ensemble de problématiques. En effet, il faut par exemple, gérer le transfert de données, l'hétérogénéité des machines cibles, l'incompatibilité entre produits, etc. La section suivante définit la problématique de notre travail et ses objectifs.

2. PROBLEMATIQUE ET OBJECTIFS

Le déploiement à grande échelle ne considère plus une seule machine cible, mais un grand nombre, qui ne sont plus gérable « manuellement ». Les différentes cibles peuvent alors avoir des caractéristiques et besoins différents. Par exemple, certaines machines peuvent fonctionner sous *Windows*, d'autres sous *Linux*, certaines peuvent avoir une faible capacité mémoire, etc. Les besoins peuvent aussi être différents : par exemple, le service de comptabilité d'une entreprise aura besoin des logiciels appropriés, alors que d'autres départements de l'entreprise n'en auront pas l'utilité. Le déploiement à grande échelle doit donc prendre en compte l'hétérogénéité des machines cibles et des besoins, tout en facilitant le travail de l'administrateur qui ne peut pas se déplacer sur chaque machine à chaque déploiement.

Le déploiement à grande échelle implique le besoin d'un *environnement pour le déploiement automatisé d'applications à grande échelle*. Un tel environnement permet aux entreprises de déployer un produit sur un ensemble de machines. Une version du produit est alors choisie pour chaque machine cible, en fonction des contraintes imposées par le produit (système d'exploitation, capacité mémoire, espace disque, …) et des caractéristiques de la machine.

Pour réaliser un tel environnement, un ensemble d'informations et de concepts doivent être modélisés. Ces informations concernent, par exemple, la description du produit à déployer : les propriétés qui le décrivent et les contraintes qu'il impose. Il s'agit aussi d'informations sur l'environnement cible. Cet environnement peut être divisé en deux parties :

- une partie décrit les machines cibles, avec un ensemble de propriétés (logicielles et/ou matérielles) et leur état ou contexte de déploiement (produits déjà déployés).
- une partie décrit la structure de l'entreprise et regroupe les différentes machines en groupes. Ces groupes peuvent être formés selon l'organisation administrative de l'entreprise (chef de services, service comptabilité, …), selon des caractéristiques communes aux machines (machines fonctionnant sous un système d'exploitation spécifique, machines ayant peu de ressources, …) ou selon tout autre critère.

De plus, pour pouvoir déployer tout type de produit (applications de type COTS « composant sur étagère », applications à base de composants, …), il faut pouvoir fournir une modélisation de ces concepts la plus générique possible. Pour cela, nous proposons de définir un *méta-modèle pour le déploiement*. A partir de ce méta-modèle, divers modèles conformes peuvent être décrits, permettant ainsi de spécifier les concepts nécessaires à un type de produit spécifique ou à un environnement cible particulier.

En outre, le déploiement à grande échelle laisse place à différents degrés de flexibilité. Par exemple, plusieurs versions d'un produit peuvent être compatibles avec la configuration d'une machine cible, plusieurs ordonnancements peuvent être possibles pour déployer un produit sur un ensemble de machines, etc. Il faut alors gérer ces différents degrés de libertés en fonction des besoins et/ou préférences de l'entreprise, cible de déploiement. Pour cela, nous proposons de définir des *stratégies de déploiement*. Permettre à l'administrateur (« déployeur ») de l'entreprise de définir ses propres stratégies de déploiement lui assure alors la possibilité de pouvoir réaliser exactement le déploiement qu'il souhaite, et de manière automatisé.

L'ensemble des informations nécessaires au déploiement (méta-modèle) et les stratégies sont ensuite utilisées pour construire un *plan de déploiement*. Un plan de déploiement définit les étapes à réaliser pour déployer un ensemble de produits (versions d'un produit) sur un ensemble de machines cibles. L'ensemble des informations étant modélisé, il est alors possible de le générer automatiquement. Un plan de déploiement peut encore contenir certains degrés de libertés, si l'utilisateur a décidé de réaliser les choix éventuels le plus tard possible. Par exemple, l'ordonnancement entre les différents déploiements unitaires (une version d'un produit sur une machine) peut être déterminé au moment de l'exécution du plan, en fonction de la charge de chaque machine cible.

Notre objectif est de fournir un environnement pour le déploiement automatisé d'applications à grande échelle. Cet environnement doit être le plus générique possible, pour pouvoir déployer tout type de produit sur n'importe quel environnement cible. Il doit aussi prendre en compte les stratégies de l'entreprise pour construire un plan de déploiement qui permet de déployer un ensemble d'applications sur un ensemble de machines. La section suivante décrit la démarche que nous avons suivie pour atteindre ces objectifs.

3. DEMARCHE SUIVIE

Pour automatiser le déploiement, un ensemble d'informations doit être modélisé. Pour cela, nous avons choisi une approche basée sur l'ingénierie dirigée par les modèles [Fav04]. Nous définissons alors un méta-modèle de déploiement le plus générique possible. Un tel méta-modèle permet d'abstraire tous les concepts nécessaires au déploiement. A partir de celui-ci, il est ainsi possible de déployer tout type de produits (applications type COTS, applications à base de composants, ...) dans n'importe quel environnement cible (structure de l'entreprise et cibles terminales (PC, portable, …) de déploiement).

Un tel méta-modèle permet ensuite de définir des modèles conformes. Chacun des modèles peut ensuite être plus spécifique. Une telle approche permet d'être générique au niveau des données de déploiement (produit, cible). Elle est ainsi réutilisable dans plusieurs contextes de déploiement.

Pour suivre une telle approche, nous utilisons deux technologies, développées par notre équipe de recherche : les procédés et les fédérations. Les procédés permettent de décrire les

différentes étapes à réaliser pour effectuer un déploiement. Les fédérations permettent de créer des applications en suivant une approche d'ingénierie dirigée par les modèles. Elles permettent aussi de réutiliser des concepts déjà définis et de les composer avec d'autres pour créer l'environnement souhaité.

Nous avons choisi de gérer la flexibilité liée au déploiement en utilisant des stratégies. Il faut alors définir précisément ce qu'est une stratégie, quelles entités sont concernées, comment doit être appliquée une stratégie, etc. Définir des stratégies entraînent aussi des cas complexes où des stratégies peuvent interagir entre elles. Il est alors important d'analyser et d'étudier ces différents cas.

Il est aussi important de ne pas se limiter à quelques stratégies pré-définies dans l'environnement. En effet, la plupart des outils existants incluent des stratégies « codées en dur ». Il est alors impossible pour un utilisateur de définir ses propres stratégies de déploiement. Nous mettons donc en œuvre les mécanismes qui lui permettent de définir n'importe quel type de stratégies à partir de quelques caractéristiques identifiées.

Enfin, pour l'implémentation, nous avons procédé en deux étapes. La première étape nous a permis de mettre en œuvre un environnement pour le déploiement automatisé, en utilisant les technologies que nous avons citées précédemment. La seconde étape, nous a permis d'ajouter la dimension d'entreprise à l'approche. Ainsi, il est devenu possible de déployer des produits sur un ensemble de machines à partir de l'application de stratégies de déploiement. Cette étape est réalisée en construisant un plan de déploiement, qui permet de déployer un ensemble de produits (versions de produits) sur un ensemble de machines, en respectant les contraintes (stratégies) imposées par l'entreprise.

L'ensemble de notre démarche est décrite dans ce document. La section suivante décrit son organisation.

4. ORGANISATION DU DOCUMENT

Ce document est organisé en trois grandes parties. La première concerne le cadre conceptuel du travail et comporte les deux chapitres suivants :

- le **chapitre 2** présente les différents points qui motivent ce travail. Ce chapitre définit aussi l'ensemble des concepts de base utilisés dans le domaine du déploiement.
- le **chapitre 3** propose un état de l'art et de la pratique. Différentes approches sont utilisées pour présenter des aspects différents du déploiement. Nous nous sommes focalisés sur les aspects liés à notre problématique et à nos objectifs.

La deuxième partie concerne notre contribution et comporte les trois chapitres suivants :

- le **chapitre 4** définit notre méta-modèle de déploiement. Il s'agit de définir un méta-modèle générique pour pouvoir permettre le déploiement de tout type de produits dans n'importe quel environnement cible (entreprise et machines).
- le **chapitre 5** décrit le travail de conception, basé sur le méta-modèle du chapitre précédent. Notre approche utilise deux technologies qui sont développées dans notre équipe de recherche : les fédérations et les procédés.

- le **chapitre 6** met l'accent sur les stratégies de déploiement. Il définit des exemples de stratégies et les classe en différentes catégories et types. Il discute aussi des différentes interactions qu'il peut exister entre stratégies.

La troisième partie concerne la réalisation et l'évaluation du travail. Elle comporte un seul chapitre :

- le **chapitre 7** présente les prototypes et expérimentations qui ont été effectués au cours de notre travail. La réalisation s'est effectuée en deux étapes. La première étape nous a permis de définir un environnement pour le déploiement automatisé, qui a subi une expérimentation en milieu industriel. La seconde étape de la réalisation implémente un environnement orienté stratégies, qui permet de prendre en compte la structure de l'entreprise et ses stratégies de déploiement.

Enfin, le document se termine par le **chapitre 8** qui propose une synthèse des travaux et présente les futures directions de travail.

CHAPITRE 2 :
LE DEPLOIEMENT :
CONCEPTS ET CONTEXTE DE TRAVAIL

Préambule :

Le déploiement est une étape du cycle de vie du logiciel qui a pris de plus en plus d'importance avec le développement d'Internet. Contrairement aux procédés traditionnels, il est maintenant possible de déployer une application chez un client, à distance et automatiquement depuis le producteur. Les grandes entreprises sont devenues soucieuses de gérer cette étape automatiquement. Ce chapitre présente les besoins et concepts liés au déploiement.

---

1. INTRODUCTION

Comme le montre la Figure 2.1, le cycle de vie du logiciel [JMS+92, Rol97] est composé schématiquement des différentes étapes suivantes :

- L'**analyse** consiste à analyser les différents besoins que l'application devra remplir.
- La **spécification** permet de formuler les fonctionnalités de l'application.
- La **conception** permet de modéliser les données utilisées.
- La **réalisation** est l'étape d'implémentation d'une solution.
- Les **tests** permettent de valider la réalisation.
- Enfin, le **déploiement** permet de gérer l'application depuis son installation jusqu'à sa désinstallation complète d'un site. Cette étape a pris de l'importance, notamment, avec le développement d'Internet. En effet, il est désormais possible de déployer une application chez un client, directement depuis le producteur, alors que les procédés plus traditionnels exigeaient un support, tel qu'un CD par exemple.

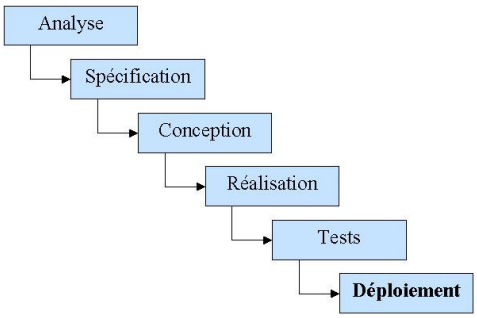

Figure 2.1 Cycle de vie du logiciel

Dans ce chapitre, nous présentons en détail cette dernière activité du cycle de vie du logiciel et ses besoins. Pour cela, nous commençons par mettre en parallèle le déploiement « standard » avec le cas du déploiement à grande échelle dans le cadre des entreprises. Puis, nous présentons les concepts principaux du déploiement en général, puisqu'ils constituent la base des notions utilisées. Pour cela, nous listons les différentes étapes d'un déploiement qui mènent à la définition du cycle de vie du déploiement. Enfin, avant de conclure, nous montrons comment ces concepts s'adaptent à notre contexte de travail sur le déploiement à grande échelle. Ainsi, nous déterminons une architecture fonctionnelle à utiliser et les concepts spécifiques au déploiement.

2. DEPLOIEMENT VS DEPLOIEMENT A GRANDE ECHELLE

Le processus de déploiement arrive en fin du cycle de vie du logiciel. Il débute lorsque le produit (ou application) est validé par le producteur, et continue jusqu'à la désinstallation des

sites des clients. Entre ces deux étapes, il faut considérer plusieurs activités, comme l'installation ou la mise à jour. En considérant toutes ces activités, on peut définir un cycle de vie du déploiement (paragraphe 4). L'ensemble des activités ainsi définies doit être pris en compte pour gérer un ensemble de machines d'utilisateurs.

Dans les grandes entreprises, le déploiement prend beaucoup d'importance [Hal99]. En effet, celles-ci ont un nombre de machines considérable à gérer. Il est donc intéressant de leur fournir un support automatisé pour l'ensemble des activités de déploiement [LBC02]. Par extension, un tel support peut servir à effectuer toute l'administration du parc informatique de l'entreprise.

Pour les entreprises, il est aussi nécessaire de prendre en compte toute leur organisation (Figure 2.2) pour gérer le déploiement. En effet, les utilisateurs, selon leur fonction, n'ont pas les mêmes besoins logiciels. Ainsi, un responsable du personnel n'a pas les mêmes besoins qu'un développeur informatique. Différents « groupes » de machines peuvent ainsi être formés, certains ayant des intersections avec d'autres. Un support pour le déploiement doit pouvoir prendre en compte les différences de besoins et la structuration des machines de l'entreprise.

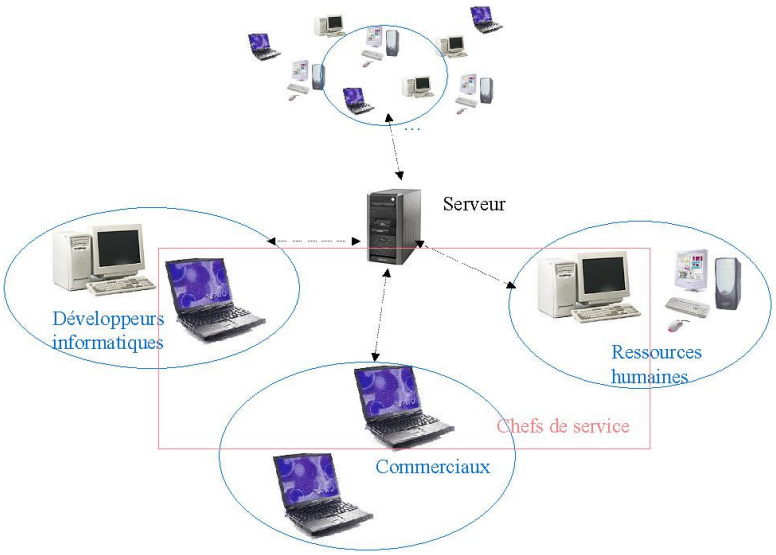

Figure 2.2 Exemple d'organisation d'une entreprise

Par exemple, sur la Figure 2.2, quatre groupes sont visibles : les *développeurs informatiques*, le services des *ressources humaines*, les *commerciaux* et l'ensemble des *chefs de service*. Nous pouvons noter que certaines machines appartiennent à plusieurs groupes (*ressources humaines* et *chefs de service*, par exemple).

En outre, l'organisation de l'entreprise peut aussi permettre de définir des stratégies de déploiement [HHW99-c]. Par exemple, il faut pouvoir exprimer qu'un déploiement doit se faire pour les machines du service informatique avant les machines du service des ressources humaines. Un autre exemple serait que l'ensemble des machines des développeurs informatiques doit avoir la même version d'un compilateur.

Pour ne pas gêner les utilisateurs de l'entreprise dans leur travail, il est aussi intéressant de pouvoir programmer le déploiement d'applications à des moments de la journée où le travail des employés n'est pas perturbé. Pour de telles activités, un environnement automatisé pour le déploiement, et supportant des stratégies, est indispensable.

Les entreprises étant souvent composées d'un ensemble de machines hétérogènes, un environnement de déploiement doit prendre en compte ce genre d'information. En effet, il doit pouvoir gérer tout type de machines : PC, serveurs, ordinateurs portables, … Il doit aussi prendre en compte les différences logicielles pour permettre de déployer des applications sous divers systèmes d'exploitation (Windows, Linux, …).

Un support automatisé pour le déploiement permet aussi aux entreprises d'avoir un contrôle complet sur l'ensemble des machines. En effet, dans cette situation, les utilisateurs n'installent plus eux même les applications dont ils ont besoin (via Internet ou via un CD d'installation). Tout déploiement passe alors par l'environnement de déploiement, géré par les administrateurs et/ou les responsables. Ainsi, à tout moment, il est possible de connaître quel application est installée sur quelle machine et de conserver la cohérence de chaque machine ou ensemble de machines. On peut ainsi assurer, par exemple, qu'un groupe de développeurs utilisent la même version d'un compilateur.

Pour simplifier la tâche des administrateurs, les producteurs d'applications fournissent un ensemble de *packages* à l'entreprise. Un package regroupe l'ensemble des données concernant l'application et les informations utiles pour son déploiement. Un package peut servir à une ou plusieurs activités de déploiement. Il contient toute l'information nécessaire au déploiement et la manière de l'utiliser. C'est ensuite à l'entreprise d'utiliser ces données comme elle le souhaite.

Pour définir quelles problématiques entrent en jeu lors du déploiement, dans le paragraphe suivant, nous citons les différentes étapes nécessaires au déploiement d'une application.

3. LES ETAPES D'UN DEPLOIEMENT

Le processus de déploiement permet d'installer et de gérer l'application [CE99]. Ainsi, on gère l'application depuis son installation jusqu'à sa désinstallation complète des sites utilisateurs. Dans cette partie, nous commençons par présenter les concepts essentiels à la compréhension du déploiement. Ainsi, nous décrivons les activités à effectuer pour réaliser un déploiement.

3.1. Composition d'une application

Lorsqu'un client est intéressé par une application (ou logiciel), il « contacte » le producteur ou un fournisseur. La distinction entre le producteur et le fournisseur peut se faire :

- Le **producteur** crée l'application sous plusieurs versions.
- Le **fournisseur** fournit un ensemble de *packages*. Chaque package contient une version d'une application, créée auparavant par le producteur.

Dans la suite du document, nous attribuons ces deux fonctions à la même personne. C'est pourquoi, par simplification, le discours fait plus souvent référence au terme de

« producteur ». Eventuellement, le client pourra être informé lorsque la première version (ou une nouvelle version) de l'application sera validée.

Une application est souvent formée de différents composants pouvant être choisis par le client, directement ou indirectement (en choisissant les services souhaités, par exemple). Certains composants sont dépendants d'autres composants ou applications. Toutes ces informations sont fournies au client, par l'intermédiaire du modèle de produit [Les00], défini dans la section 6.3. Dès que le client a choisi les composants de l'application, le producteur prépare tous les composants nécessaires. Ces composants, ainsi que le modèle de produit, sont ensuite conditionnés sous forme de package pour être transférés chez le client.

La Figure 2.3 illustre deux configurations possibles d'une application (produit). Un client choisissant la configuration 1 pourra faire de la conception graphique en 3 dimensions et de l'animation vidéo sur une machine Windows. Un client optant pour la configuration 2 fera de la conception en deux dimensions sous Linux avec une aide au format texte.

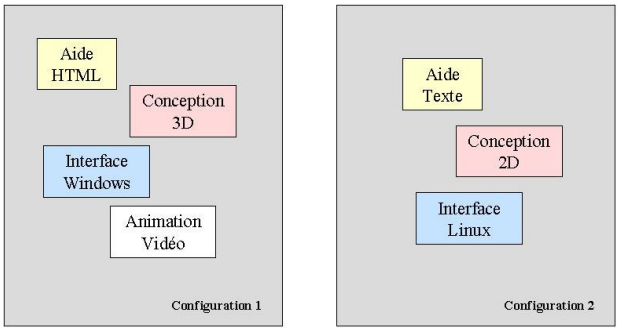

Figure 2.3 Exemples de composition d'un produit

Il faut aussi noter que le producteur peut fournir plusieurs versions de l'application ayant les fonctionnalités souhaitées par le client : par exemple, deux versions implémentées dans deux langages différents. Le client reçoit alors plusieurs versions d'une même application en fonction de ses besoins. Ces versions sont équivalentes, mais n'ont pas les mêmes caractéristiques (langage, système d'exploitation, …). L'entreprise cliente peut ensuite choisir la (les) version(s) qui correspondent le mieux à chacune des machines de son parc informatique.

Le client doit alors sélectionner quelle(s) version(s) il désire déployer sur ses machines. Ce choix dépend d'un ensemble de stratégies de déploiement que l'entreprise applique et des configurations matérielles et logicielles des machines ciblées pour le déploiement. Ainsi, avec une stratégie, l'entreprise peut, par exemple, favoriser le déploiement des unités implémentées en java ou forcer le déploiement de la même version sur toutes ses machines. Dans ce deuxième cas, il faut trouver une version qui soit compatible avec les configurations de toutes les machines.

3.2. Transfert de l'application

Pour transférer l'application du producteur vers le client, il y a deux modes (Figure 2.4) :

- **La technique du « push »** : c'est le producteur qui décide d'envoyer l'application au client lorsqu'il le désire. Dans ce cas, le client s'est inscrit auparavant à une liste de diffusion.
- **La technique du « pull »** : c'est le client qui décide de l'instant où il voudra transférer l'application sur son site. Ceci lui permet aussi une plus grande sécurité sur son site. Cette technique s'appuie sur le concept de serveur d'applications, qui met un ensemble de versions d'applications à disposition des clients.

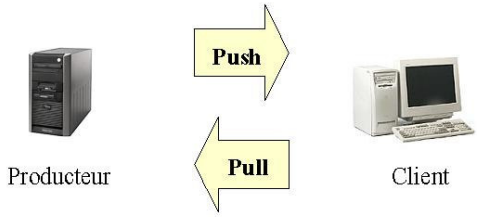

Figure 2.4 Modes de transfert d'un produit

Nous avons présenté ces techniques de transfert, comme une relation entre le producteur et un client. Mais cette relation peut aussi s'appliquer entre le producteur et une entreprise, qui gère ensuite elle-même les transferts vers les machines clientes concernées. Cette relation peut donc aussi avoir lieu entre un serveur d'application d'une entreprise et les machines clientes de cette entreprise. Nous verrons par la suite (paragraphe 5) comment se situe le niveau entreprise.

Une fois l'application transférée, le client, s'il le souhaite, peut personnaliser son application.

3.3. Personnalisation de l'application

Le producteur doit aussi fournir un environnement de travail, pour que les propres extensions ou le paramétrage de l'entreprise cliente puissent être intégrées à l'application. Lors de telles changements, de nouvelles contraintes ou dépendances peuvent être ajoutées. Dans ce cas, le modèle de produit est modifié pour prendre en compte les nouvelles informations.

De plus, l'entreprise peut personnaliser l'application pour chaque membre du personnel. Les besoins de chaque employé sont définis dans le modèle d'entreprise. Ainsi, chaque employé ou groupe d'employés peut avoir une configuration particulière et spécifique à ses besoins.

L'entreprise choisit ensuite une stratégie de déploiement. Par exemple, une stratégie, dite du « big-bang », consiste à déployer l'application vers tous les utilisateurs en même temps. D'autres stratégies moins globales sont possibles selon les besoins de l'entreprise (établir un ordre partiel de déploiement, quand déployer les dépendances d'un produit, …). Après ces étapes, on sait quel utilisateur recevra quelle version de l'application et à quel moment. Il ne reste alors plus qu'à déployer l'application concrètement sur chaque machine.

3.4. Déploiement concret

Le déploiement concret est réalisé selon la stratégie choisie au niveau entreprise. Au moment voulu, des instructions sont exécutées pour installer et configurer l'application (c'est-à-dire chacun de ses composants) que le site utilisateur final reçoit.

La configuration est spécifique à chaque utilisateur, puisqu'elle dépend, en particulier, des caractéristiques matérielles et logicielles de sa machine. Il faut donc vérifier que les contraintes et dépendances, établies lors de la sélection des composants (au niveau entreprise), sont bien vérifiées. Par exemple, si l'application à déployer a des dépendances, il faudra peut-être d'abord installer celles-ci. Toutes ces caractéristiques (logicielles et matérielles) sont conservées dans le modèle de machine (ou modèle de site) [Gom00].

Après avoir terminé les vérifications, on peut installer la bonne configuration de l'application pour chaque utilisateur. Les instructions à exécuter pour réaliser l'installation sont fournies par le producteur de l'application, puisque c'est lui qui a la connaissance de son produit (procédure de déploiement unitaire). Cependant ces instructions peuvent avoir des paramètres qui permettent de personnaliser les actions en fonction des caractéristiques du site cible (par exemple : spécification de chemins de déploiement).

3.5. Gestion de l'application

L'application, une fois installée, doit être gérée durant toute la durée où elle est présente sur le site utilisateur, c'est-à-dire jusqu'à sa désinstallation. Ainsi, on doit, par exemple, pouvoir le mettre à jour lorsque cela est nécessaire. En fait, un ensemble d'activités peut être réalisées : on parle de cycle de vie du déploiement.

4. LE CYCLE DE VIE DU DEPLOIEMENT

Dans la suite des concepts liés au déploiement, nous présentons le cycle de vie du déploiement. La Figure 2.5 illustre la partie déploiement du cycle de vie du logiciel [CFH+98]. Le déploiement prend en compte toutes les étapes liées à la vie du logiciel depuis sa validation par le producteur jusqu'à sa désinstallation des sites clients. Ces étapes consistent notamment à installer l'application, la mettre à jour, la reconfigurer ou la désinstaller. Chacune des étapes de ce cycle de vie est détaillée dans les paragraphes suivants.

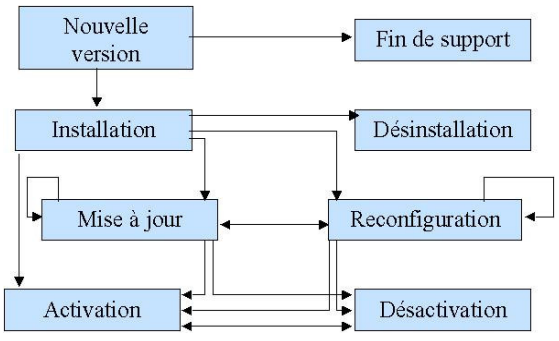

Figure 2.5 Cycle de vie du déploiement d'un logiciel

Dans cette partie, nous nous focalisons sur le déploiement unitaire d'un producteur vers un client. Le producteur/fournisseur fournit des versions d'applications aux clients, disponibles sur des serveurs d'applications. Chaque client est représenté par une seule machine cible. Nous verrons dans la partie suivante comment passer au déploiement à grande échelle, dans le contexte des entreprises, en définissant une architecture fonctionnelle.

4.1. Nouvelle version de l'application

Lorsque le producteur publie une nouvelle application, il en informe ses clients. Ceux-ci peuvent ensuite sélectionner et configurer une version qui leur convient [CE00]. Pour cela, le client indique quelles fonctionnalités il souhaite. Il peut aussi imposer quelques contraintes par rapport à la configuration de ses machines. Par exemple, il peut demander une version pour Windows.

En fonction de ces choix, le producteur sélectionne les composants nécessaires pour construire l'application pour son client. L'ensemble de ces composants forme l'application destinée au client. Le producteur doit aussi indiquer comment déployer ces unités. Pour cela, il fournit aussi un plan de déploiement indiquant les étapes à suivre (copie ou exécution de fichiers, par exemple). Le plan de déploiement peut être destiné à réaliser chacune des activités du déploiement (installation, mise à jour), selon la demande du client.

L'ensemble de ces informations (produit, description et plan de déploiement) est ensuite packagé. Ce package résultant peut ensuite être transféré vers l'entreprise l'ayant choisi.

4.2. Installation

L'activité d'installation consiste à installer physiquement l'application sur le site client. Pour cela, il faut tout d'abord transférer le package (obtenu chez le producteur) vers le site client concerné. Le transfert peut s'effectuer selon l'une des deux techniques vues précédemment (push ou pull).

Une fois le package sur le site cible, il faut récupérer l'information qu'il contient. Ensuite, il faut instancier le plan de déploiement en fonction des caractéristiques de la machine si cela est nécessaire. Il s'agit de donner des valeurs précises aux paramètres du plan, en fonction de la configuration de la machine cible. Par exemple, l'administrateur détermine où réaliser l'installation (chemins de déploiement).

Ensuite, il n'y a plus qu'à lancer l'exécution du plan de déploiement. Pour réaliser l'installation, il peut être nécessaire d'utiliser des outils disponibles sur la machine cible : par exemple, un outil permettant de copier des fichiers.

L'utilisation ou l'installation de l'application peut aussi avoir des dépendances. On dit qu'une application dépend d'une autre, lorsqu'elle l'utilise pour pouvoir fonctionner correctement. L'administrateur de l'entreprise peut exiger de résoudre les dépendances avant de commencer l'installation. Il faut alors vérifier si les applications nécessaires sont déjà installés sur le site client. Si c'est le cas, l'installation peut débuter. Si ce n'est pas le cas, il faut résoudre les dépendances, en installant aussi les applications concernées.

Avant de pouvoir exécuter et utiliser l'application installée, il faut ensuite passer par l'étape d'activation.

4.3. Activation et désactivation

L'activation permet de préparer l'environnement dans lequel l'application devra s'exécuter [Han99] et d'instancier l'application. Il s'agit par exemple de créer des instances de base de données. Cette étape peut ne concerner que la machine cible, mais elle peut aussi concerner un ensemble de machines dans le cas des applications distribuées. Dans ce deuxième cas, l'activation peut consister à créer l'ensemble des passerelles nécessaires pour la communication des machines entre elles. Après l'activation, l'application est prête à être exécutée et à fonctionner correctement.

La désactivation est l'opération inverse de l'activation. Elle consiste à désactiver l'ensemble des unités de l'application pour pouvoir effectuer des modifications sur celui-ci. La désactivation est donc effectuée avant une mise à jour, une reconfiguration ou une désinstallation. Cette activité n'est utilisée que pour des modifications statiques de l'application, puisque les modifications dynamiques interdisent l'arrêt complet (ou la désactivation) de l'application.

Après avoir effectué une désactivation dans le but de réaliser une mise à jour ou une reconfiguration, l'application doit être activée une nouvelle fois avant de pouvoir s'exécuter à nouveau. L'activation (ré-activation) a évidemment lieu après avoir réalisé les modifications nécessaires.

4.4. Mise à jour

La mise à jour a lieu lorsque le producteur propose une nouvelle version de son application. La mise à jour peut s'effectuer à la demande du client ou à l'initiative du producteur si le client s'est inscrit à un service de mise à jour. En général, un producteur fournit une nouvelle version d'une application pour corriger une erreur de fonctionnement, pour ajouter de nouvelles fonctionnalités ou améliorer les performances.

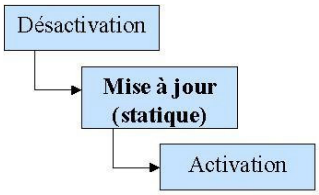

Figure 2.6 Activité de mise à jour

La mise à jour statique nécessite de désactiver l'application sur le site client (Figure 2.6). Il faut ensuite récupérer les informations nécessaires chez le producteur et effectuer les modifications indiquées. Ces modifications peuvent être des modifications mineures : la copie d'une nouvelle DLL, par exemple [And00]. Mais elles peuvent aussi être plus importantes et impliquer la désinstallation de l'ancienne application pour installer la nouvelle version. Dans les deux cas, il faut suivre le plan de déploiement indiqué par le producteur et fourni (avec les données nécessaires) dans le package de la nouvelle version de l'application. Après avoir effectué les modifications indiquées par le producteur, l'application peut à nouveau être activée (Figure 2.6).

La mise à jour peut aussi s'effectuer de façon dynamique : c'est l'activité d'adaptation dynamique (non mentionnée dans la Figure 2.5) [KB03]. La mise à jour s'impose lorsque l'arrêt de l'application pour effectuer des modifications n'est pas souhaitable [MH01]. Ceci est vrai pour certains types d'applications, comme :

- les applications **à haute disponibilité** comme les services de commerces électroniques et les applications bancaires,
- les applications utilisées dans les **domaines critiques et temps réel** [Fab76, SF93], comme les applications de contrôle industriel et les services de télécommunications,
- les applications **embarquées** [AM00] dans des équipements distants ou mobiles,
- les applications **complexes**, dont l'arrêt est coûteux.

Pour ce type d'applications, la mise a jour doit avoir lieu dynamiquement, au cours de l'exécution [KBC02]. Cette activité nécessite un important travail de préparation et des technologies avancées pour mettre en œuvre les modifications à apporter. Cette activité fait l'objet d'études et de diverses propositions de solutions [Ket04, LH05, PBJ97], qui ne seront pas exposées ici.

4.5. Reconfiguration

L'activité de reconfiguration est assez proche de l'activité de mise à jour, puisqu'il s'agit d'apporter des modifications à l'application. Comme pour la mise à jour, il peut donc être nécessaire de désactiver l'application avant la reconfiguration et de le ré-activer après.

A la différence de la mise à jour, il n'est pas nécessaire de contacter le producteur pour reconfigurer une application. Il s'agit simplement de fournir une configuration plus adaptée au site client avec les informations déjà connues. La reconfiguration peut ainsi se produire dans les cas suivants :

- Les **caractéristiques** matérielles et/ou logicielles du site utilisateurs sont modifiées.
- Des **extensions** sont faites au niveau de l'entreprise, ce qui modifie l'application à ce niveau. Si l'utilisateur est concerné par ces changements, la configuration de l'application pour sa machine doit être modifiée.

Une reconfiguration peut donc avoir lieu, par exemple, lorsque l'utilisateur ajoute de la mémoire à sa machine, lorsqu'il change de carte vidéo, ou lorsque l'entreprise ajoute un service à l'application.

Pour qu'une reconfiguration se produise, il faut que l'utilisateur ou l'administrateur en aient fait la demande. Cette demande peut se faire soit par l'utilisateur lui-même (s'il modifie la configuration de sa machine), soit par l'administrateur de l'entreprise (si l'application est modifiée au niveau entreprise).

Comme le producteur n'est pas contacté à nouveau lors d'une reconfiguration, cela implique que l'information et les données nécessaires étaient contenues dans le package de l'installation initiale. Cela sous-entend aussi que ce package a été conservé par le client.

4.6. Désinstallation

L'activité de désinstallation consiste à retirer l'application du site client. Pour cela, on suit, à nouveau, les instructions (plan de déploiement) indiquées par le producteur, afin de retirer toute trace de l'application sur la machine.

Avant de réaliser une telle action, il faut prendre en compte les utilisations de l'application et ses dépendances. En effet, il faut vérifier si les dépendances de l'application doivent aussi être désinstallées (en même temps que l'application elle-même) ou non. Il faut aussi faire attention à ne pas retirer une application qui est utilisée par d'autres, pour que celles-ci puissent continuer à fonctionner correctement.

4.7. Fin de support

Lorsque l'application est devenue obsolète, mais qu'il est encore utilisé chez des clients, c'est l'activité de fin de support. Dans ce cas, le producteur ne fournit plus de nouvelles versions, ni de services (assistance téléphonique, par exemple) concernant le produit. Cette activité arrête l'évolution du produit dans le temps.

Après cette activité, l'application ne peut donc plus être mise à jour chez les clients l'utilisant. Elle n'est plus disponible, non plus, pour être installée chez de nouveaux clients. Les clients chez lesquels l'application est déjà installée peuvent, eux, quand même continuer à l'utiliser.

5. ARCHITECTURE FONCTIONNELLE

Dans les parties précédentes, nous avons présenté les concepts spécifiques au déploiement. Nous montrons maintenant comment ces concepts s'adaptent à notre contexte de travail, en passant au cas du déploiement à grande échelle. Ce cas concerne les entreprises qui doivent gérer un grand nombre de machines cibles. Les sections suivantes présentent notre vision du déploiement à grande échelle et les travaux déjà effectués dans notre équipe de recherche. Dans ces sections, des termes plus génériques sont utilisés : nous parlons ainsi plus facilement de produit que d'application, puisque nous nous intéressons au déploiement au sens large du terme (déploiement d'applications ou de composants d'applications).

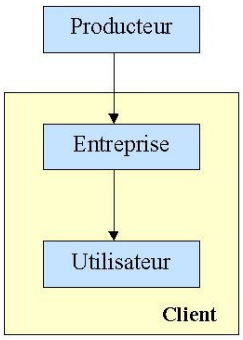

Figure 2.7 Les trois acteurs du déploiement

Pour répondre aux divers besoins du déploiement, nous définissons une architecture fonctionnelle d'un environnement de déploiement. Comme l'indique la Figure 2.7, plusieurs acteurs entrent en jeu lors du déploiement d'un produit (ou application) :

- le **producteur**/fournisseur, chargé de livrer un produit à ses clients,
- l'**entreprise**, responsable de la préparation du déploiement sur chaque site,
- l'**utilisateur**, sur le site duquel le produit est déployé physiquement.

Chacun d'eux a un rôle à remplir en réalisant différentes activités.

Dans une entreprise, il faut installer un produit sur un grand nombre de machines. Il faut alors gérer les configurations particulières de chaque machine cible. Pour faciliter cette tâche, on ajoute un niveau entre le niveau producteur (développement du produit) et le niveau utilisateur (installation du produit) : il s'agit du niveau entreprise [LBC00].

Pour réaliser le processus de déploiement, il faut effectuer des activités sur chacun des trois niveaux. L'ensemble de ces activités permet de personnaliser le déploiement des produits par rapport aux propositions du producteur, aux contraintes imposées par l'entreprise et aux caractéristiques de l'utilisateur.

5.1. Niveau producteur

Au niveau producteur, tout est fait pour préparer l'envoi des produits aux clients. Il s'agit ici du commencement de l'activité de déploiement. Il faut alors envoyer le produit, les outils servant à son extension dans l'environnement de l'entreprise, des outils de tests, et toute l'information nécessaire pour son installation et sa gestion sur les sites clients tout au long du cycle de vie du déploiement. L'information descriptive du produit est contenue dans le modèle de produit (ou modèle d'application). L'information relative à son déploiement se trouve dans le plan de déploiement unitaire, spécifique à une version d'un produit. L'ensemble de ces éléments forme l'application (ou produit) producteur.

A chaque version de produit est associé un modèle de produit, qui décrit l'architecture de la version du produit. Par exemple, il contient les options du produit et les dépendances entre produits. Le modèle de produit contient aussi un ensemble d'information caractérisant le produit et pouvant servir au client. On peut trouver comme information : le langage d'implémentation du produit, le(s) système(s) d'exploitation pour le(s)quel(s) il est conçu, etc. Certaines des informations du modèle de produit peuvent être commune à plusieurs versions du produit.

A chaque version de produit est aussi associé un plan de déploiement unitaire. Ce plan peut être spécifique à une version du produit ou être commun à plusieurs versions. Ce plan donne le mode d'emploi permettant de déployer la version concernée. Par exemple, il indique quels fichiers copier ou exécuter et dans quel ordre le faire. Ce plan est ensuite instancié par le client avec des informations spécifiques (comme des chemins de déploiement…).

Dès qu'une nouvelle version d'un produit (éventuellement la première version) est disponible, le producteur l'annonce aux entreprises clientes. Le producteur a la possibilité de configurer le produit avant de l'envoyer au client (niveau entreprise) : après avoir choisi toutes les fonctionnalités, les composants du produit correspondants sont sélectionnés et le produit est créé. Plusieurs versions de celui-ci peuvent correspondre aux besoins et préférences de

l'entreprise. Dans ce cas, chaque version est créée et packagée, puis l'ensemble des packages est transmis au niveau entreprise.

La Figure 2.8 schématise le travail du producteur. Le client lui donne ses contraintes de sélection et de configuration. En fonction de cela, le producteur sélectionne la(les) version(s) appropriée(s) à son client, ainsi que le modèle de produit et le plan de déploiement correspondants à chaque version. Ces informations sont ensuite packagées pour être destinées au client. Un package concerne une seule version d'un produit (unité de déploiement) et son information.

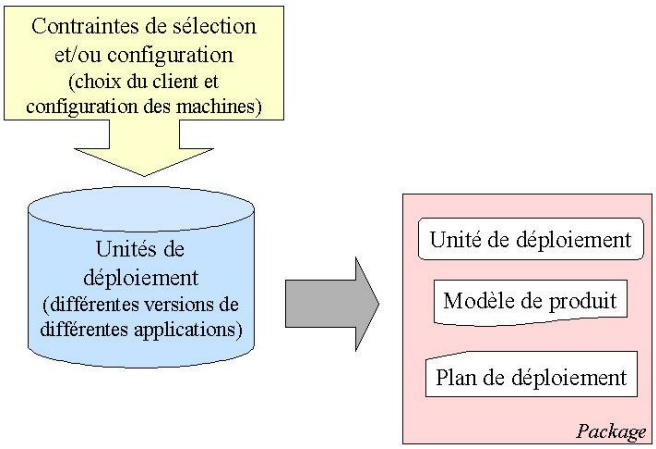

Figure 2.8 Création d'un package par le producteur pour le client

A la fin des activités réalisées par le producteur, les versions de l'*application (ou produit) producteur* sont transférées vers le site de l'entreprise cliente. Chacune contient tous les éléments cités précédemment, mais aussi, éventuellement, un environnement de travail permettant d'ajouter des composants ou des fonctionnalités au produit.

5.2. Niveau entreprise

Ce niveau va permettre de préparer le déploiement concret vers les machines des utilisateurs. Les propres extensions de l'entreprise peuvent être ajoutées à l'application producteur afin de la personnaliser : c'est l'étape d'assemblage. Ces extensions sont ajoutées avec l'aide des outils de construction et de tests fournis par le producteur. Le nouveau produit ainsi créé est appelée *application (ou produit) entreprise*.

L'entreprise peut avoir de nombreuses versions d'un produit, récupérées chez un fournisseur, sous forme packagée. En effet, si plusieurs versions correspondaient à ses besoins, elles ont toutes été transférées sur les serveurs d'applications de l'entreprise. Les modifications effectuées au niveau entreprise peuvent concerner chaque version du produit : l'application entreprise est donc aussi disponible en plusieurs versions. Chaque modèle de produit peut être modifié si le changement réalisé par les extensions ajoute des contraintes ou dépendances. Les modèles de produit de chaque version (unité de déploiement) de l'application entreprise sont utilisés à l'étape suivante.

Ensuite, il faut fixer où, quand et comment le déploiement du produit est réalisé. Différentes versions du produit peuvent être déployées. Il s'agit d'établir un lien entre le produit de

l'entreprise (disponible en plusieurs versions ou unités de déploiement) et son organisation. L'organisation de l'entreprise est décrite dans un modèle d'entreprise associé à des stratégies de déploiement (ensemble de contraintes, de politiques de contrôle du déploiement et la méthode à employer). Ces stratégies peuvent exprimer toutes sortes de choses qui intéressent l'entreprise pour le déploiement dans son environnement. Elles permettent, par exemple, de favoriser un type de produit (implémentation en java par exemple), d'ordonner les différentes étapes d'un déploiement (sur la machine M1, puis sur la machine M2 par exemple), de traiter les erreurs qui surviennent au cours du déploiement (retour-arrière, continuer, …), d'imposer un certain niveau de sécurité, de planifier le déploiement pendant la nuit, etc. Le modèle de l'entreprise décrit l'organisation de l'entreprise en termes d'équipes, d'agents humains et de rôles. Il est important de prendre en compte ces informations, car elles peuvent influencer le processus de déploiement.

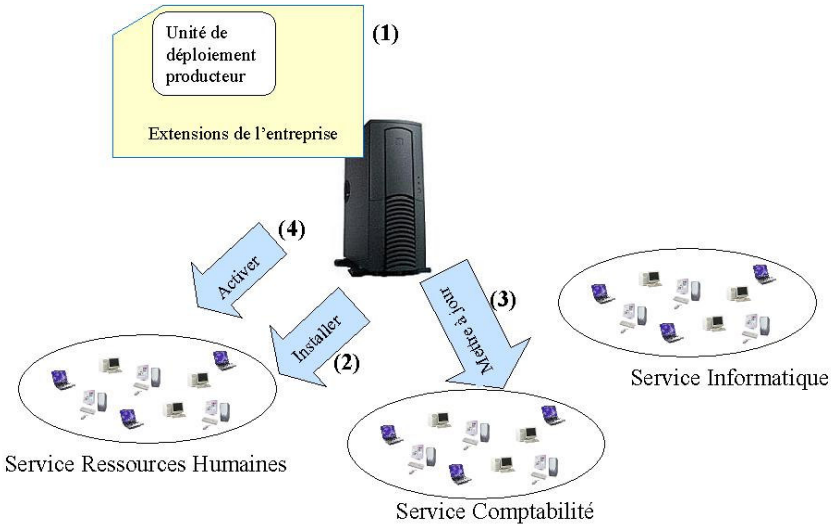

Figure 2.9 Définition d'un plan de déploiement pour l'entreprise

La Figure 2.9 indique un exemple des étapes à réaliser au niveau entreprise. Ici, il s'agit de déployer une version précise d'une application sur toutes les machines de l'entreprise. Certaines machines utilisent déjà une version antérieure de l'application : dans ce cas, il s'agit de faire une mise à jour avec la nouvelle version. D'autres machines n'ont jamais utiliser cette application : il faut réaliser une première installation de la nouvelle version. Pour cela, il faut réaliser les étapes suivantes :

- ajout des **extensions** de l'entreprise à l'unité (version de l'application) de déploiement (repère **(1)**),
- définition d'un **plan de déploiement** à réaliser dans l'ordre suivant : installer (cf. paragraphe 4.2) sur les machines du service des ressources humaines (repère **(2)**), mettre à jour (cf. paragraphe 4.4) les machines du service de comptabilité (repère **(3)**), puis activer (cf. paragraphe 4.3) l'unité pour les machines des ressources humaines (repère **(4)**).

A la fin de cette activité, on obtient le plan de déploiement, qui indique quand un utilisateur aura telle version du produit, en fonction de sa position dans l'entreprise. Il reste alors à

déployer concrètement l'application entreprise sur les machines clientes en fonction des informations fournies par le plan de déploiement.

5.3. Niveau utilisateur

A ce niveau, le modèle de machine donne les informations sur la configuration logicielle et matérielle du site client. Le déploiement concret s'effectue en plusieurs étapes : la configuration, la mise à disposition, le transfert et l'installation.

La configuration consiste à choisir les options convenables pour l'installation en fonction de l'information fournie par le modèle de machine. Lors de la mise à disposition, le produit est prêt à être déployé sur la machine de l'utilisateur. Le transfert est l'étape où le produit est transporté sur le site client (généralement sous forme de package). Enfin, le produit est installé, en suivant les indications données par le plan de déploiement.

Après l'installation du produit, on doit aussi considérer toutes les autres activités du cycle de vie du déploiement, comme la mise à jour, l'adaptation ou la reconfiguration. Pour gérer ces activités, il peut, à nouveau, être nécessaire de contacter le producteur, afin d'obtenir de nouvelles informations, données ou plans de déploiement.

5.4. Organisation physique de l'architecture fonctionnelle

Chacun des niveaux (producteur, entreprise et utilisateur) de l'architecture fonctionnelle a un rôle bien défini. Il faut ensuite déterminer les machines physiques chargées de remplir ces rôles. La Figure 2.10 fait le lien entre les niveaux de l'architecture fonctionnelle et l'architecture physique utilisée pour le déploiement.

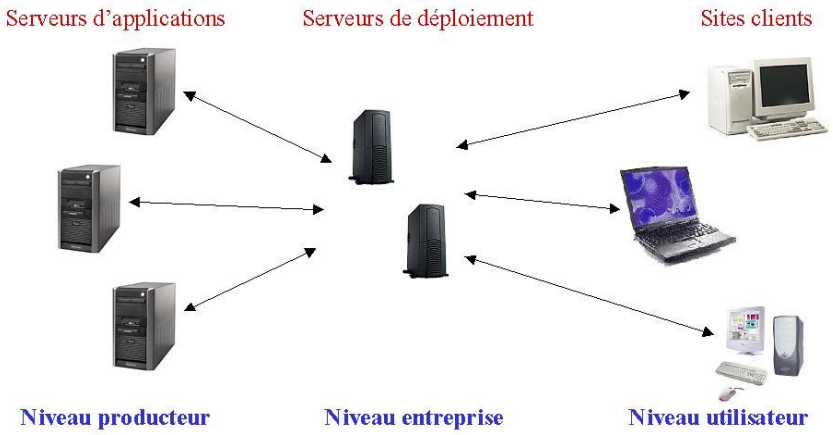

Figure 2.10 Architecture physique

Le rôle du producteur (fournisseur) est rempli par un (ou plusieurs) **serveurs d'applications**. Ils sont chargés de mettre des unités de déploiement à la disposition des entreprises clientes. Un ou plusieurs **serveurs de déploiement** effectuent les tâches du niveau entreprise. Ils sont chargés de piloter et superviser le déploiement des unités à distance. Enfin, les **sites clients** sont les machines des utilisateurs sur lesquelles seront déployés (puis utilisés) les produits.

Il faut aussi noter que les serveurs d'applications (niveau producteur) peuvent aussi faire partie de l'entreprise. En effet, d'une part, l'entreprise peut développer elle-même ses propres produits. D'autre part, l'entreprise peut commander des unités de déploiement à des producteurs extérieurs et les mettre à disposition de ses utilisateurs sur ses propres serveurs d'applications. L'entreprise crée ainsi sa propre architecture fonctionnelle de déploiement, même si toutes les machines physiques appartiennent à l'entreprise.

6. LES CONCEPTS DE BASE

Chacun des niveaux (producteur, entreprise et utilisateur) que nous venons de définir a des fonctions précises. Pour remplir ces fonctions au mieux, un ensemble d'informations doit être connu et pris en compte. Pour structurer ces informations, nous définissons plusieurs modèles :

- le modèle d'**entreprise**,
- le modèle de **machine** (ou modèle de site),
- le modèle de **produit** (ou modèle d'application),
- le plan de **déploiement** (ou modèle de déploiement).

Ces différents éléments constituent les concepts de base liés au déploiement. Ils sont présentés brièvement ci-dessous. Ils seront détaillés dans les chapitres suivants

6.1. Modèle d'entreprise

Le modèle d'entreprise représente la structure complète de l'entreprise où sont déployés les produits logiciels. Ce modèle contient toute l'organisation de l'entreprise. Ainsi, par exemple, il peut contenir la représentation des différentes équipes de travail, les différents services de l'entreprise (ressources humaines, comptabilité, …), les divers utilisateurs, mais aussi les machines appartenant à l'entreprise. Ce modèle peut contenir un ensemble d'informations qui sont ou non utilisées pour le déploiement, comme des politiques (de déploiement ou autre), des informations sur le statut des membres de l'entreprise, etc. Ce modèle peut aussi être utilisé dans d'autres contextes (gestion du personnel, par exemple), c'est pourquoi toute l'information contenue n'est pas forcément utilisée pour le déploiement.

Dans le cadre du déploiement, il est important de retenir la structuration des machines en différents groupes et, éventuellement, sous-groupes. Ces groupes peuvent être réalisés selon des critères hiérarchique de l'entreprise (équipes, membres d'une équipe, …) ou selon des critères plus techniques (machines utilisant Windows, machines ayant un faible espace disque, machines dont la langue de l'utilisateur est le français, …).

Le modèle d'entreprise permet aussi d'associer les différentes stratégies (ou politiques) de déploiement qui sont utilisées. Ces stratégies établissent, par exemple, un ordre de déploiement spécifique ou une préférence pour des produits java. Ces stratégies sont utilisées pour définir un plan de déploiement global à l'entreprise ou à un groupe de machines de l'entreprise.

6.2. Modèle de machine

Le modèle de machine décrit la configuration matérielle et logicielle des sites clients. Chaque site possède son propre modèle de machine. La configuration matérielle comprend toutes les caractéristiques matérielles de la machine comme, par exemple, le type de la carte graphique, celui du processeur, la capacité mémoire, etc.

La configuration logicielle comprend toutes les caractéristiques logicielles, qui définissent notamment le système d'exploitation, mais aussi l'état du site. Celui-ci indique quels produits logiciels sont déjà installés sur le site client.

Le modèle de site constitue le contexte de déploiement. Ainsi, les caractéristiques (matérielles ou logicielles) sont utilisées pour vérifier si un produit peut être déployé ou non sur la machine, en fonction des contraintes qu'il impose. L'état du site sert à identifier quels produits sont déjà installés pour ne pas déployer plusieurs fois les mêmes. Par exemple, lorsqu'un produit a des dépendances ou des incompatibilités, il faut vérifier si les produits concernés sont sur la machine ou non.

6.3. Modèle de produit

Le modèle de produit est défini (et fourni) par le producteur. Il est spécifique à une version d'un produit (unité de déploiement) et contient l'ensemble des informations qui le concernent. Ainsi, les caractéristiques du produit (caractéristiques de l'application ou caractéristiques spécifiques à la version) y sont indiquées, comme le langage d'implémentation par exemple. Un ensemble de contraintes définit aussi les caractéristiques que doit avoir une machine cible potentielle pour le bon fonctionnement de la version de l'application, comme par exemple le système d'exploitation pour lequel le produit est conçu.

Le modèle de produit contient aussi les dépendances et/ou incompatibilités qu'il existe entre ce produit et d'autres. Le modèle de produit peut être modifié au niveau entreprise, si celle-ci ajoute ses propres extensions.

Dans le cadre du déploiement, le modèle de produit est utilisé pour sélectionner quelle(s) version(s) déployer sur les machines de l'entreprise. En effet, il est important de déployer un produit qui fonctionne correctement sur une machine, et donc de vérifier si la machine remplit les contraintes qui sont imposées par celui-ci.

Au moment du déploiement, il faut aussi vérifier si les produits dépendants et/ou en conflit sont déjà déployés sur les machines cibles. En fonction de cela, les dépendances peuvent aussi être déployées. Pour les produits en conflit (incompatibilité), le produit déjà déployé peut être supprimé ou le déploiement du nouveau produit peut être arrêté : cela dépend de la politique appliquée par l'entreprise.

De plus, les différentes propriétés du produit pourront être utilisées pour répondre aux besoins et politiques de déploiement de l'entreprise. Ainsi, par exemple, une entreprise peut choisir de ne déployer que des produits implémentés en java.

6.4. Plan de déploiement

Le plan de déploiement unitaire décrit comment déployer une version d'un produit (unité de déploiement) sur un site client. Le producteur fournit un plan de déploiement unitaire avec chacun de ses produits. Celui-ci donne le mode d'emploi permettant de déployer son produit.

Au niveau entreprise, un plan de déploiement global est établi : il permet de déployer un ensemble de produits sur un ensemble de machines de l'entreprise. Il est construit à partir des plans de déploiement unitaires fournis par le producteur et des stratégies appliquées par l'entreprise.

Un plan de déploiement contient les instructions à réaliser pour effectuer un déploiement unitaire (une unité de déploiement sur une machine) ou non (plusieurs unités sur plusieurs machines). Ces instructions peuvent être exprimées et interprétées de différentes manières. Par exemple, un plan de déploiement peut être représentées par une suite de tâches à exécuter, un procédé, ou à l'aide d'un langage à interpréter. Cette représentation fait partie de la personnalisation du déploiement selon les objectifs de l'entreprise.

7. NOTRE APPROCHE DU DEPLOIEMENT A GRANDE ECHELLE

Notre approche a pour objectif de fournir un environnement de déploiement automatisé qui permette de gérer tout le cycle de vie du déploiement, voire toute l'administration des machines d'une entreprise. Dans cette thèse, nous nous focalisons plus particulièrement sur l'installation des produits fournis par le producteur sous forme packagée.

Il s'agit alors de pouvoir sélectionner une (des) version(s) du produit (parmi la liste des versions disponibles) adaptées aux machines cibles. La version déployée peut être la même sur toutes les machines ou être différente : cela dépend des opérations de déploiement (stratégies de l'entreprise).

Notre approche vise à définir un méta-modèle de déploiement qui permette ensuite d'établir les différents modèles (entreprise, machine, produit, plan de déploiement) qui sont utilisés dans le déploiement. Une telle approche permet d'être générique au niveau du type des produits à déployer, comme au niveau de l'environnement cible de déploiement.

L'automatisation du déploiement implique l'utilisation de l'information contenue dans les différents modèles. Mais il est aussi important de prendre en compte les contraintes qui peuvent être imposées par l'entreprise : ordre de déploiement, préférence pour un type de produit, traitement des erreurs, etc. C'est pourquoi nous avons défini des stratégies de déploiement.

Les stratégies de déploiement permettent de personnaliser le déploiement en fonction de l'entreprise cible. Elles peuvent être utilisées à différents moment du déploiement : pour la sélection, pour l'ordonnancement des différentes étapes, etc.

L'application des stratégies permet ensuite de créer un plan de déploiement. Le plan de déploiement regroupe toutes les étapes à réaliser pour effectuer le déploiement de *n* applications sur *p* machines. Il s'agit en fait d'ordonnancer des déploiements unitaires (une application sur une machine). Le plan de déploiement peut aussi comporter différents degrés

de liberté. Par exemple, l'ordre de déploiement ne peut être que partiel : des stratégies peuvent alors être utilisées pour déterminer l'ordre précis au moment où l'exécution du plan débute.

Notre approche vise à fournir les moyens aux entreprises, de définir tout type de stratégies. Celles-ci sont ensuite prises en compte pour automatiser le déploiement, tout en considérant les contraintes imposées par l'entreprise elle-même. L'approche est réalisée en deux temps :

- la définition et la mise en oeuvre d'un **méta-modèle de déploiement** pour fournir un support automatisé pour le déploiement d'applications à grande échelle, et,
- la définition et la mise en œuvre de **stratégies de déploiement** pour gérer la flexibilité du déploiement pour chaque entreprise.

8. SYNTHESE

Le déploiement exige la connaissance et l'utilisation de domaines variés, qui interviennent à des instants différents. Pour répondre à la problématique du déploiement, il faut, tout d'abord, être capable d'assurer toutes les activités du cycle de vie du déploiement, c'est-à-dire l'installation , la désinstallation, la mise à jour, la reconfiguration, l'activation et la désactivation. L'adaptation dynamique (mise à jour ou reconfiguration en cours d'exécution) peut aussi être ajoutée à cette liste pour certains types d'applications.

Ainsi, un environnement de déploiement doit permettre de gérer l'installation d'un produit sur une machine, la mise à jour d'un autre produit sur d'autres machines, etc. Il doit gérer toutes ses étapes séquentiellement et/ou parallèlement selon les contraintes imposées par l'entreprise, et les réaliser sans gêner le travail des utilisateurs, c'est-à-dire sans bloquer l'utilisation des machines cibles ou en imposant un minimum de perturbations.

De plus, il faut prendre en compte l'hétérogénéité des machines cibles. En effet, chaque machine peut avoir une configuration logicielle et matérielle différente. Il faut donc être capable d'établir un déploiement sur tous les types de machines en prenant en compte les caractéristiques de chacune. Par exemple, il faut être capable de déployer un produit aussi bien sur une machine *Windows* que *Linux*, dans des versions éventuellement différentes.

L'environnement des machines est aussi amené à être modifié constamment, soit en termes d'applications (nouveaux produits installés, anciens supprimés, …), soit en termes de matériels (changement de carte vidéo, ajout de mémoire, …). Il faut être capable de répondre à ces nouvelles configurations, en adaptant le déploiement à la situation (par exemple, en reconfigurant un produit).

En outre, pour assurer que les produits déployés fonctionneront correctement, il faut étudier les contraintes qu'ils imposent, et les relations entre les différents produits et savoir résoudre les dépendances entre eux. Il faut aussi éventuellement savoir résoudre des problèmes d'incompatibilité entre les produits. Les produits pouvant être des applications client/serveur ou des applications distribuées, il faut aussi gérer les relations entre les machines.

Enfin, il faut pouvoir répondre aux besoins des entreprises, qui tentent de trouver des solutions ad hoc [CVC+04] pour leurs problèmes de déploiement. Pour cela, nous utilisons une architecture à trois niveaux. Chacun des niveaux a des fonctions précises. Le niveau producteur est chargé de mettre à disposition les produits avec l'information qui les concerne.

Le niveau entreprise prépare le déploiement vers les machines cibles. Le niveau utilisateur est le niveau où les produits sont déployés concrètement, puis utilisés.

Ces trois niveaux utilisent un ensemble d'informations structurées en plusieurs modèles de données. Le modèle de produit décrit le produit à déployer. Le modèle d'entreprise représente la structure de l'entreprise et les politiques de déploiement à appliquer. Le plan de déploiement indique comment déployer un produit. Enfin, le modèle de machine caractérise la configuration d'un site client.

De plus, un ensemble de stratégies permet de décrire quelles politiques de déploiement sont appliquées par l'entreprise. Avec ces stratégies, l'entreprise peut alors accordé plus d'importance à un type de produit (par exemple, les produits implémentés en java), établir un ordre dans les diverses étapes du déploiement à réaliser, sécuriser un déploiement (pour éviter de déployer des produits non désirés), etc.

Notre approche vise à fournir un environnement de déploiement automatisé où les contraintes imposées par l'entreprise sont prises en compte par l'intermédiaire de stratégies de déploiement. Pour être le plus générique possible, notre environnement est basé sur un méta-modèle de déploiement.

CHAPITRE 3 : ETAT DE L'ART ET DE LA PRATIQUE

Préambule :

Ce chapitre présente quelques uns des modèles qui ont été définis pour le déploiement. Il présente aussi un ensemble d'environnements et d'outils existants dans le monde académique ou industriel. L'objectif est de réaliser un bilan des concepts à définir pour le déploiement et des techniques qui peuvent être utilisées pour répondre à la problématique du déploiement.

---

1. INTRODUCTION

Différentes approches ont pu être définies et utilisées pour tenter de répondre à la problématique du déploiement. Parmi ces approches, les dernières à apparaître sont celles qui utilisent une approche à base de modèles. Ces approches sont à mettre en relation avec les approches MDE (*Model Driven Engineering*) [Fav04, Omg03-a], qui prennent de plus en plus d'importance dans le monde de la recherche. Ce chapitre commence donc par présenter quelques modèles définis pour le déploiement.

En utilisant ces modèles, dont certains deviennent plus ou moins des standards dans le domaine, ou en définissant leurs propres modèles plus spécifiques, les chercheurs proposent des environnements de déploiement. Quelques uns d'entre eux sont donc présentés dans la suite du chapitre.

Ensuite, des outils commerciaux sont aussi décrits. Une quantité assez importante d'outils existe, mais la plupart d'entre eux reste plus ou moins spécifique quand à leurs objectifs. En effet, nous avons vu dans le chapitre précédent que le déploiement met en jeu beaucoup de problématiques, et les outils existants répondent rarement à toutes.

Enfin, nous terminons ce chapitre par une synthèse de l'ensemble des informations énoncées, que ce soit celles liées aux modèles, celles liées aux environnements de recherche ou celles liées aux outils commerciaux.

Dans ce chapitre, nous nous limitons aux approches utilisant des modèles ou aux approches ayant un lien avec le déploiement à grande échelle et l'utilisation de stratégies de déploiement. Ces deux thèmes sont les objectifs principaux de cette thèse, c'est pourquoi nous nous focalisons sur eux.

2. DES MODELES GENERIQUES POUR LE DEPLOIEMENT

Plusieurs modèles pour le déploiement ont été définis. Nous en présentons ici trois : celui proposé par l'OMG, celui associé à Java et un autre (pour les applications à base de composants) issu du monde de la recherche.

2.1. Spécification pour le déploiement et la configuration d'applications distribuées à base de composants, proposée par l'OMG

L'OMG propose une spécification pour le déploiement et la configuration d'applications distribuées à base de composants [Omg04]. Cette spécification est en partie basée sur l'ingénierie dirigée par les modèles (MDE). En effet elle définit un modèle indépendant de la plate-forme (PIM) à spécialiser en un modèle spécifique à une plate-forme (PSM), selon un modèle à composants particulier. Pour présenter ce modèle, nous décrivons d'abord la décomposition du modèle défini, puis les acteurs qui interviennent au cours du déploiement.

2.1.1. Décomposition du modèle

Trois domaines différents sont utilisés dans la spécification de l'OMG :

- les applications à base de composants,
- l'environnement cible,
- le procédé de déploiement.

Les **applications à base de composants** doivent être décrites précisément pour permettre leur déploiement sur une cible.

- Un composant peut avoir des implémentations qui sont soit du code compilé, soit un assemblage d'autres composants.
- Un assemblage de composants est un ensemble de composants et leurs interconnexions.
- Une application à déployer est considérée comme un simple composant : il n'y a pas d'interface spécifique pour les composants qui peuvent être déployés comme des applications.
- Les composants sont distribués en packages.
- Un package est un ensemble de méta-données et des modules de code compilé correspondants. Il peut contenir plusieurs implémentations d'une interface de composant : ceci permet de choisir l'implémentation correspondant le mieux à l'environnement cible.
- De plus, une implémentation de composant impose des contraintes qui devront être satisfaites par les propriétés du système cible. Les contraintes d'un assemblage de composants sont celles des sous-composants et celles de leurs connexions.

L'**environnement cible** décrit les caractéristiques de l'environnement où les composants sont déployés, puis utilisés.

- Un environnement cible est appelé domaine.
- Un domaine est composé de nœuds, d'interconnexions et de passerelles.
- Un nœud a une capacité de calcul et représente la cible où seront exécutées les implémentations de composants.
- Les interconnexions fournissent une connexion partagée directe entre les nœuds (un câble Ethernet, par exemple).
- Les passerelles représentent les routeurs et ponts (*switches*).
- Les nœuds, les interconnexions et les passerelles ont des caractéristiques, des ressources et des capacités. Par exemple, un nœud peut avoir un système d'exploitation ou une capacité mémoire, une interconnexion peut avoir une certaine bande-passante.
- Le logiciel qui supporte l'exécution des composants sur un nœud particulier (environnement d'exécution) doit pouvoir être implémenté indépendamment du service de déploiement.

Le **procédé de déploiement** permet de décrire les différentes phases du déploiement. Le procédé débute après le développement, le packaging et la publication du logiciel, et après son

acquisition par son propriétaire (le « déployeur »). Le procédé de déploiement prend en compte les étapes suivantes :

- Lors des **pré-conditions pour le procédé de déploiement**, le package de l'application à déployer est publié et rendu disponible au « déployeur ».
- L'**installation** apporte le package dans un endroit où l'application sera configurée en appliquant des politiques de déploiement.
- La **configuration** permet de configurer les différents composants pour leur future exécution.
- L'étape de **planning** définit comment et où l'application s'exécutera dans l'environnement cible.
- La **préparation** réalise le travail nécessaire pour que l'environnement cible soit prêt à exécuter l'application.
- Le **lancement** amène l'application dans un état d'exécution, en utilisant les ressources nécessaires.

Le modèle indépendant de la plate-forme est défini en prenant en compte l'ensemble de ces informations.

2.1.2. Modèle indépendant de la plate-forme (PIM)

Le modèle est sectionné en plusieurs grandes parties :

- modèle de données et modèle de gestion de composant,
- modèle de données et modèle de gestion de cible de déploiement,
- modèle de données et modèle de gestion d'exécution.

Les modèles de données, descriptifs, sont ainsi séparés des modèles de gestion, permettant d'effectuer une série d'opérations sur les données.

2.1.2.1. Modèle de composant

Le modèle à composants est basé sur une vision composant « standard ». Ainsi, un composant a une interface et peut être connecté à d'autres composants par l'intermédiaire de ports. Un composant peut avoir une implémentation monolithique (un seul artefact). Un composant peut aussi être un composant composite, c'est-à-dire implémenté par un assemblage de composants (ensemble de sous-composants interconnectés).

Un package de composant contient plusieurs implémentations du même composant. Ainsi, il est possible de distribuer un ensemble d'implémentations avec des propriétés ou des hiérarchies différentes, dans un même package. Ces packages sont installés dans un dépôt (*repository*) où ils peuvent être configurés avant le déploiement.

Les composants et packages sont aussi associés à un ensemble de descriptions (description d'implémentation, d'assemblage, …) et de besoins pour exprimer comment les ressources vont être utilisées.

2.1.2.2. Modèle de cible de déploiement

Le modèle de cible décrit et gère l'information du domaine où sont déployées des applications. Un domaine représente l'environnement cible complet. Il est composé d'un ensemble de nœuds interconnectés entre eux et communicants par des passerelles.

Les passerelles représentent la description d'un chemin de communication directe entre les nœuds. Les interconnexions décrivent une connexion directe partagée entre des nœuds. Elles permettent aux composants instanciés sur les nœuds de communiquer avec des composants sur d'autres nœuds.

Des nœuds peuvent partager des ressources, contenues dans le domaine, entre eux. Enfin, un nœud, une interconnexion ou une passerelle peut avoir des caractéristiques (capacité de calcul, …), appelées ressources, à l'intérieur de l'environnement cible.

2.1.2.3. Modèle d'exécution

Avant le déploiement, des décisions doivent être prises pour sélectionner les implémentations des composants à déployer et déterminer où chacune d'elles doit être déployée. Toute l'information sur le déploiement d'une application est collectée dans un plan de déploiement. Le plan de déploiement contient l'information sur les artefacts faisant partie du déploiement, comment créer les instances de composant à partir des artefacts, où les instancier, l'information sur les connexions entre eux, …

2.1.3. Acteurs

Pour définir l'ensemble des modèles présentés ci-dessus, plusieurs acteurs entrent en jeu :

- Le « **spécifieur** » spécifie l'interface et le contrat fonctionnel pour les implémentations de composant.
- L' « **implémenteur** » crée les implémentations concrètes des composants, avec leurs méta-données. Cet acteur peut avoir deux rôles différents : le développeur et l'assembleur. Le développeur crée les implémentations monolithiques, puis l'assembleur crée les implémentations des composants composites.
- Le « **packageur** » crée les packages des implémentations de composants.
- L' « **administrateur de domaine** » prépare l'information sur l'environnement cible.

De plus, trois acteurs sont impliqués dans le déploiement d'une application :

- L' « **administrateur de dépôt »** reçoit des packages de composant et les installe dans un dépôt local.
- Le « **planificateur** » prend des décisions sur le déploiement en se basant sur les capacités de la cible et les besoins du composant. Pour cela, il fait correspondre les besoins de l'implémentation avec les ressources disponibles de la cible et crée un plan de déploiement spécifique.
- L' « **exécutant** » utilise le plan de déploiement pour exécuter le déploiement et instancier l'application.

Le planificateur peut définir un ensemble de plans de déploiement valides, c'est-à-dire un ensemble de plans de déploiement qui décrivent un déploiement valide de l'application dans le domaine cible. Certains de ces plans pouvant être plus appropriés à un contexte donné, la sélection de l'un d'entre eux dépend de l'implémentation finale. Un algorithme est défini pour déterminer les plans valides pour le déploiement d'une application dans un environnement cible.

2.1.4. Conclusion

Le PIM de la spécification a été utilisé pour définir un PSM pour CCM. Ceci montre la bonne applicabilité de la spécification pour un modèle à composants. Cependant, il semble plus difficile de l'adapter à des applications commerciales (type COTS), dont on ne connaît pas forcément une description aussi détaillée, en termes de composants.

De plus, les contraintes de déploiement ne sont exprimées que selon les besoins des applications. Les sites clients ne sont définis que par des ressources et des propriétés qui permettent de remplir ou non les contraintes imposées par les applications. Il serait intéressant d'ajouter des stratégies de déploiement à ce niveau, pour définir, par exemple, une préférence pour un certain type d'implémentation de composant.

La spécification propose un autre point intéressant : le plan de déploiement. Celui-ci permet de rassembler l'ensemble de l'information nécessaire au déploiement. Il décrit où et comment déployer l'ensemble des implémentations de composants de l'application dans un environnement donné. L'automatisation de la génération de ce plan est essentielle dans un contexte de déploiement à grande échelle. De plus, la création de plusieurs plans de déploiement laisse un certain degré de liberté jusqu'au moment de l'exécution du déploiement. Ainsi, le déploiement peut être adapté à l'environnement cible le plus tard possible. Cependant, il pourrait aussi être intéressant d'adapter le déploiement (l'ordre des étapes, par exemple) pendant l'exécution du plan.

Enfin, il est important de préciser que ce modèle est devenu une référence en matière de déploiement, notamment grâce à l'importance du travail réalisé et de ses auteurs.

2.2. Le modèle abstrait de déploiement Java

Le modèle abstrait de déploiement Java [Jav02] se caractérise par les rôles (acteurs) qui entrent en jeu, les différentes étapes qui composent le cycle de vie du déploiement, et les autres concepts ou définitions qui formalisent les aspects du déploiement. Ces points sont définis dans les paragraphes suivants.

2.2.1. Les acteurs du modèle

Quatre acteurs (Figure 3.1) entrent en jeu dans le modèle abstrait de déploiement java. Ces acteurs remplissent chacun un rôle bien défini. L'ensemble des actions réalisées par ces acteurs permet de déployer une application.

Le producteur doit construire un produit logiciel et le préparer pour le déploiement. Il doit fournir des ressources annotées (ressources à déployer et leur description) et les politiques de déploiement. La partie producteur peut se séparer en deux acteurs :

- Le **producteur** développe le logiciel.

- Le **propriétaire du logiciel** a les droits sur le produit et décide des politiques de déploiement.

Pour simplifier le modèle, ces deux acteurs peuvent coïncider. En effet, dans la pratique, c'est souvent le même acteur qui est responsable du développement du logiciel et de ses droits.

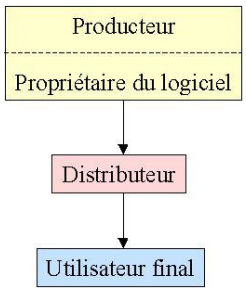

Figure 3.1 Les acteurs du modèle abstrait de déploiement Java

L'acteur suivant qui entre en jeu dans le modèle est le **distributeur**. Il est chargé de distribuer et installer correctement le logiciel sur la plate-forme cliente. Sa tâche consiste donc à exécuter correctement les politiques de déploiement sur un ensemble donné de ressources annotées pour un groupe d'utilisateurs donné.

Enfin, l'**utilisateur final** est celui qui utilise le logiciel déployé sur la plate-forme cliente. Après avoir défini les différents acteurs qui entrent en jeu dans le modèle, nous introduisons, dans la suite, les concepts de base du modèle.

2.2.2. Les concepts

Plusieurs concepts sont définis dans le modèle abstrait de déploiement Java. Les principaux sont l'AH (*Application Helper*), l'environnement client et les politiques de déploiement.

L'**AH** (*Application Helper*) est un programme qui s'exécute localement sur les plates-formes clientes. Les plates-formes clientes supportent le téléchargement, l'installation physique et les exécutions sous-jacentes des applications à déployer. L'AH peut être vu comme une interface entre le serveur de déploiement et le client (Figure 3.2).

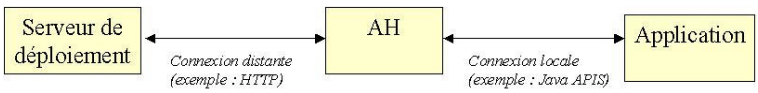

Figure 3.2 Le rôle de l'AH

En outre, l'**environnement client** est une description formelle de la configuration courante de la plate-forme cliente. Cette description rassemble aussi bien les caractéristiques matérielles que logicielles ou autres. L'environnement définit une instance courante du client. Il est constitué d'un ensemble de propriétés (couple <nom, valeur>), qui permettent notamment de détailler les points suivants :

- données sur l'utilisateur final : rôle, information sur la licence du produit, etc.,

- propriétés de connexion,
- librairies et extensions java installées,
- versions des ressources de l'application installée,
- données dépendant de la plate-forme,
- etc.

Côté serveur, un ensemble de **politiques de déploiement** définissent exactement qui doit installer quoi. Ces politiques sont fournies par le propriétaire du logiciel. Une politique est une assertion du type *IF (condition) INSTALL « nom .jar »*. Par exemple, l'assertion *IF (license== « evaluation ») INSTALL « timebomb.jar »* indique que pour une version d'évaluation un *jar* particulier doit être installé. Les politiques de déploiement peuvent contenir les informations suivantes :

- Les **politiques de connexion** ont un rôle important dans le déploiement à distance. Elles permettent, par exemple de spécifier quand établir une connexion.
- Les **politiques de distribution** sont du côté serveur du processus de déploiement et sont actionnées par le Distributeur dans la phase de gestion du serveur de déploiement. Elles permettent, par exemple, de gérer la distribution d'une application en réseau.
- Les **politiques basées sur les différentes propriétés du client** sont appliquées du côté client. Elles spécifient, par exemple, la version du JRE nécessaire à l'exécution de l'application.
- Les **politiques de mise à jour du client** permettent au propriétaire du logiciel de spécifier quand et comment la phase de mise à jour de l'application sera exécutée. Un autre attribut peut être utilisé pour définir la relation entre la politique de mise à jour et l'intervention de l'utilisateur (mise à jour exécutée explicitement ou en arrière plan, sans que l'utilisateur ne le sache).
- Les **politiques d'utilisateur final** sont conçues avec la définition des utilisateurs et l'information reliée. Elles permettent, par exemple, de vérifier les droits d'une certaine classe d'utilisateurs.

La phase de résolution permet de décider que faire en fonction de :

- ce qui est sur le client (environnement client),
- ce qui est sensé être délivré (ressources à installer),
- ce qui doit être fait (politiques de déploiement).

L'ensemble de ces concepts est ensuite utilisé au cours du cycle de vie du déploiement, géré par le modèle.

2.2.3. Etapes du cycle de vie

Lors de la publication, le distributeur doit fournir les bons éléments (composants, fichiers *.jar*), ainsi que les instructions concernant leur déploiement (politiques de déploiement), aux bons clients pour construire une application pouvant s'exécuter correctement. Pour cela, le

producteur utilise des serveurs de déploiement, qui stockent les ressources prêtes à être demandées (en mode pull) ou envoyées (en mode push) aux utilisateurs finaux. Indépendamment, le serveur de déploiement peut être configuré par l'organisation du distributeur.

Lors du téléchargement et de l'installation du logiciel, un AH est aussi téléchargé et installé. L'AH est fourni par le distributeur. Il est installé, lancé, gère l'installation de l'application, l'exécute et fournit d'autres services de déploiement à l'application en exécution (débuggage, mise à jour, …).

Comme le montre la Figure 3.3, le modèle de déploiement Java comporte les différentes phases suivantes :

- **Développement** : Le logiciel à déployer est produit et associé à des politiques de déploiement pour la publication.
- **Publication** : A la fin de cette étape, le logiciel est prêt pour l'installation pour les utilisateurs.
- **Gestion du Serveur de Déploiement (SD)** : Cette étape comporte la maintenance et la gestion des opérations (réplication, mise à jour, configuration, …) effectuées sur les serveurs de déploiement.
- **Installation de l'AH** : L'AH est installé pour la première fois sur la plate-forme cliente.
- **Exécution de l'AH** : L'AH est lancé, puis il analyse la situation du client et décide que faire : téléchargement et installation, exécution (si l'application est déjà installée), gestion de l'application (mise à jour,…), configuration, arrêter l'application en exécution. L'AH fournit ensuite différents services de déploiement (connexion au serveur de déploiement, téléchargement de ressources nécessaires, …)
- **Exécution de l'application** : Enfin, l'application est exécutée. Quand l'AH lance l'application, il doit avoir lancé la JVM, passé les paramètres…

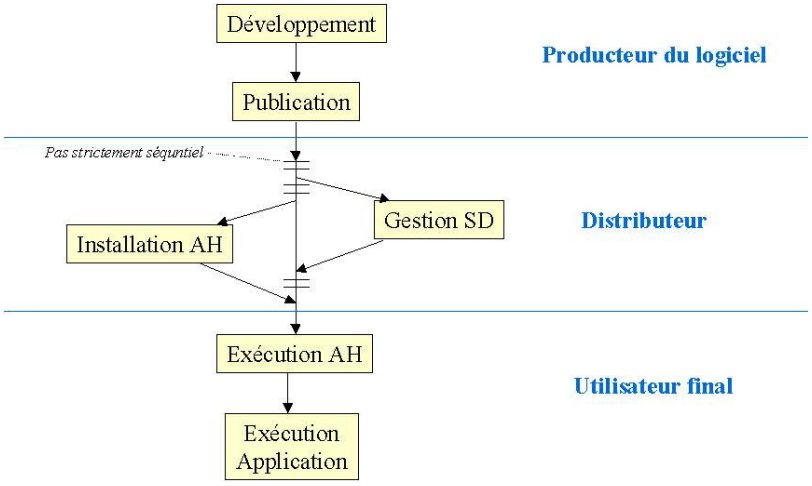

Figure 3.3 Le modèle de déploiement Java

2.2.4. Conclusion

Le modèle abstrait de déploiement Java met en place différents niveaux (Producteur, Distributeur, Utilisateur). Ces niveaux peuvent être mis en correspondance avec les niveaux conceptuels que nous avons définis précédemment (Producteur, Entreprise, Utilisateur).

En outre, les diverses activités du cycle de vie du déploiement (installation, mise à jour, etc.) ne sont pas explicitement utilisées. En effet, c'est l'assistant de déploiement (AH) qui décide quoi faire en fonction de la situation du client.

Enfin, un ensemble de politiques de déploiement peuvent être définies. Elles permettent de définir le déploiement en fonction des caractéristiques de l'application à déployer et des caractéristiques des sites clients. Cependant, la plupart sont définies sur le serveur de déploiement et ne laissent que peu de libertés au client. De plus, une vision globale (niveau entreprise) des machines est absente : il est donc difficile d'exprimer des politiques qui concernent plusieurs machines cibles.

2.3. Un modèle de déploiement conceptuel pour les applications à base de composants

Parrish et al. [PDC01] proposent un modèle conceptuel pour le déploiement d'applications à base de composants. Pour le présenter, nous commençons par décrire les concepts, puis les différentes politiques de déploiement, et une manière d'évaluer le résultat d'un déploiement.

2.3.1. Les concepts

Un **composant** est une unité qui représente une partie d'un logiciel. Un système de composants est défini comme un ensemble fini de composants Csys = {C1, C2, …, Cn} , représentant un ensemble de composants présents sur une machine particulière. Enfin, une application est une unité de comportement, remplissant des fonctionnalités précises.

Une **famille de composants** contient des composants. Tous les membres d'une famille de composants partagent la même classe générale de comportement, même s'ils n'implémentent pas forcément la même spécification fonctionnelle. Chaque composant a aussi un numéro de version (entier unique). Plusieurs composants de la même famille peuvent coexister à l'intérieur d'une configuration donnée d'une machine.

De plus, pour classer les différentes versions des composants entre elles, deux types de compatibilité sont définis :

- **Compatibilité descendante** (BC - *backward compatibility*) : si le composant remplace un composant existant plus vieux, les applications fonctionnant avec l'ancien composant, fonctionnent avec le nouveau.
- **Compatibilité ascendante** (FC – *forward compatibility*) : si le composant remplace un composant existant plus récent, les applications fonctionnant avec le composant le plus récent, fonctionnent avec le plus ancien.

2.3.2. Les politiques de déploiement

Lors de l'installation d'un nouveau composant sur une machine cliente, il se peut que le composant soit déjà installé dans une autre version. Dans ce cas, il faut prêter attention à ce que les applications qui l'utilisent fonctionnent toujours avec la version à installer. Plusieurs choix sont alors possibles pour répondre à ce problème.

Pour cela plusieurs politiques de déploiement sont définies :

- **Insertion séparée** (IS - *Insert Separatly*) : le composant à installer est ajouté comme un nouvel élément, sans remplacer d'autres composants. Le nouveau composant ne sera pas partagé par les applications existantes (car toutes celles-ci sont déjà réalisées par les composants existants).
- **Remplacement obligatoire** (RA - *Replace Always*) : le composant déjà déployé sur la cible est remplacé par l'autre composant.
- **Remplacement si nouveau** (ROIN - *Replace Only If Newer*) : le composant déjà déployé sur la cible est remplacé par l'autre composant, seulement si celui-ci a une version plus récente.
- **Aucun remplacement** (NR - *Never Replace*) : on ne fait rien : la version du composant déjà déployée est laissée intacte.

Selon la politique de déploiement qui est appliquée et en fonction des compatibilités qui existent entre les différents composants, la qualité du déploiement peut alors être évaluée.

2.3.3. Evaluation du déploiement

Pour évaluer la qualité du déploiement, deux propriétés sont utilisées :

- **Propriété de réussite** : l'application installée fonctionne correctement.
- **Propriété de sûreté** : les applications existantes continuent à fonctionner correctement après l'installation de la nouvelle application.

En fonction des politiques de déploiement et des compatibilités appliquées, le tableau de la Figure 3.4 peut être défini. Il résume les propriétés que doivent remplir les composants pour assurer les propriétés de sûreté ou de réussite au déploiement.

Politique de déploiement appliquée	Propriétés suffisantes pour garantir la sûreté	Propriétés suffisantes pour garantir la réussite	Réutilisation possible du composant
NR		FC, BC	Oui
RA	FC, BC		Oui
ROIN	BC	BC	Oui
IS			Non

Figure 3.4 Résultats de réussite et de sûreté d'un déploiement

Ainsi, par exemple, lors de l'application de la stratégie IS (insertion séparée), les propriétés de sûreté et de réussite seront assurées. Par contre, le composant ne sera pas réutilisable par d'autres applications. Comme autre exemple, nous pouvons citer qu'il faut qu'un composant installé selon la politique ROIN (remplacement si nouveau) assure une compatibilité descendante pour garantir les mêmes propriétés. En effet, avec la compatibilité descendante, les applications fonctionneront avec le composant qu'elles sont sensées utiliser (si c'est celui dont la version est la plus récente) ou avec un composant plus récent (si une version plus récente est installée). Dans ce cas, le composant pourra être utilisé par d'autres composants (composants déjà déployés ou futurs composants).

2.3.4. Conclusion

Ce modèle présente trois dimensions intéressantes. La première est qu'une installation peut être complète ou partielle, c'est-à-dire qu'un ensemble de composants installés ne représentent pas forcément une application à eux seuls.

La deuxième dimension intéressante est que plusieurs versions concurrentes sont supportées. Ainsi, plusieurs versions séparées peuvent être installées (politiques de déploiement IS) ou une seule version (autres politiques de déploiement). Enfin, une application peut choisir de partager ou non des composants avec d'autres applications. En effet, dans le cas d'une politique de déploiement IS, une installation indépendante aura lieu. Mais ce ne sera pas le cas avec les autres politiques de déploiement.

Pour terminer, il faut noter que ce modèle n'a pas encore été implémenté. Nous l'utilisons, par la suite, pour mettre en œuvre un type de stratégie de notre approche.

2.4. Discussion et synthèse sur les modèles génériques pour le déploiement

En analysant les modèles que nous venons de décrire, il est assez aisé de faire ressortir un ensemble de concepts liés au déploiement. Ainsi, nous retrouvons tout d'abord des modèles qui permettent de décrire l'ensemble de l'information nécessaire au déploiement. Quelque soit le nom que chaque modèle donne à ces concepts, nous retrouvons notamment :

- **Un modèle de produit** (ou modèle d'application, ou modèle à composants) permet de décrire ce qu'il faut déployer.
- **Un modèle de site** permet de décrire où déployer. Selon les approches, ce modèle peut comporter un ensemble de machines physiques (dans le modèle de l'OMG, un site est un ensemble de machines avec des interconnexions) ou une seule. Dans le cas où ce modèle ne concerne qu'une cible, il peut être complété par un autre modèle donnant une vision plus globale de l'organisation (**modèle d'entreprise**).

D'autres informations permettent aussi de définir comment déployer un produit. On parle alors de **plan ou de procédé de déploiement**. A nouveau, cette information peut être mentionnée à deux niveaux : soit pour un déploiement unitaire (déploiement d'une application sur une machine cible), soit pour un déploiement plus global (déploiement de *n* applications sur *p* machines).

Les différents modèles peuvent être personnalisés pour déployer un type précis d'applications sur un type précis de cibles. Par exemple, le modèle de l'OMG a été spécialisé pour déployer des applications développées selon le modèle à composants CCM. De la même manière, chaque information du modèle de site peut être personnalisée. La personnalisation du plan de déploiement, peut quant à elle s'apparenter à une technologie. Par exemple, un plan de déploiement peut revenir à exécuter un procédé ou à exécuter un ensemble de tâches.

Dans les modèles présentés, nous retrouvons aussi la structure fonctionnelle que nous avons présentée dans le chapitre précédent. En effet, les applications sont mises à disposition avec un ensemble d'informations les concernant (niveau *Producteur*), éventuellement sous des formes variées, selon le modèle. Puis, un plan de déploiement est ensuite établi pour savoir quoi déployer où (niveau *Entreprise*). Enfin, les applications sont déployées, puis exécutées sur les cibles de déploiement (niveau *Utilisateur*).

L'ensemble de ces concepts ayant été définis dans le chapitre précédent, nous ne nous attardons pas plus ici. La suite de ce chapitre nous montre comment des modèles peuvent être utilisés dans diverses approches.

Parmi les modèles que nous avons présentés ici, nous retenons plusieurs notions à utiliser dans notre approche. Le modèle abstrait de déploiement Java nous fournit l'ensemble des étapes à réaliser pour le déploiement, et la structure fonctionnelle à trois niveaux. La spécification pour le déploiement proposée par l'OMG amène la notion de plan de déploiement. Notre approche vise à construire un plan de déploiement de manière automatique et à l'exécuter ensuite. De plus, nous utilisons aussi un ensemble de concepts, issus de ces divers modèles, pour définir notre méta-modèle de déploiement (contraintes sur les composants, par exemple).

Enfin, le dernier modèle que nous avons présenté nous fournit un bon exemple de stratégies de déploiement qui peuvent être utilisées pour l'installation ou la mise à jour de composants sur un site client.

Cependant, aucun des modèles ne fournit de réel support pour le déploiement au sein d'une entreprise ayant un grand nombre de machines à gérer. Il est nécessaire d'ajouter un concept de stratégie de déploiement. Celui-ci permet de personnaliser le déploiement à grande échelle, en fonction de contraintes imposées par l'entreprise.

3. DES ENVIRONNEMENTS DE DEPLOIEMENT ISSUS DE LA RECHERCHE

L'état de l'art concernant les travaux académiques est bien illustré par l'ensemble des travaux réalisés par les partenaires du projet ITEA OSMOSE [Osm05, Gdf05]. Dans cette partie, nous avons choisi de présenter deux d'entre eux : CADeComp réalisé à l'INT (Institut National des Télécommunications) et RESOLVIT proposé par des universitaires espagnols. Auparavant, nous présentons l'un des premiers travaux réalisés dans le cadre du déploiement : Software Dock.

3.1. Software Dock

Software Dock [Hal99] est un outil universitaire qui permet d'installer et de gérer des applications de type client/serveur. Il introduit la notion de famille de logiciels, représentant un ensemble de configurations ou de versions du logiciel. Software Dock permet de décrire les différents niveaux (producteur, entreprise, utilisateur) et les configurations logicielles grâce aux fichiers DSD (*Deployable Software Description*). Ces deux concepts sont décrits dans les deux parties suivantes.

3.1.1. Différents niveaux

Les Docks [HHW99-a] permettent de modéliser les clients et les serveurs. A chaque Dock est associé une base de données (« registre ») et des agents. Les agents sont chargés de modifier la base de données. Chacune de ces modifications entraîne des événements, auxquels les agents peuvent souscrire. Il y a deux types d'agents :

- Les **agents externes** peuvent migrer vers d'autres *Docks* (par exemple, un agent d'installation).
- Les **agents internes** restent fixes à un *Dock* (par exemple, l'agent de site).

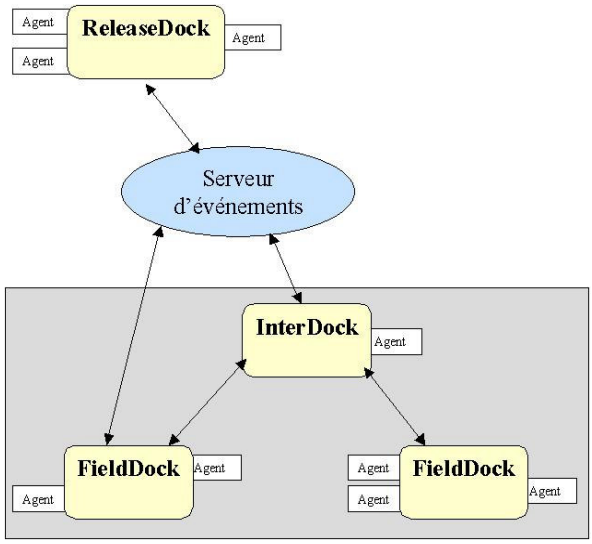

Figure 3.5 Architecture de Software Dock

Les Docks [HHH+97] servent à modéliser les différents niveaux du déploiement décrits dans la partie précédente. Le *ReleaseDock* correspond au producteur, *l'InterDock* à l'entreprise, et, le *FieldDock* à l'utilisateur final. La Figure 3.5 schématise l'architecture de Software Dock :

- Le ***FieldDock*** se trouve sur le site client et offre des informations sur l'environnement matériel et logiciel du site. Il y en a un par site consommateur. Ses agents (par exemple, l'agent de site) gèrent les modifications apportées à l'environnement.
- Le ***ReleaseDock*** réside sur le site du producteur et offre des informations sur les logiciels disponibles pour le déploiement, pouvant être modifiées via une

interface utilisateur. Il possède les agents d'installation et de mise à jour, utilisés lors du déploiement. Lorsqu'un client choisit un logiciel, un agent est envoyé sur le site client. Cet agent communique avec le *FieldDock* du consommateur et obtient l'information sur la configuration appropriée. Puis il localise les composants nécessaires dans le *ReleaseDock* et les installe sur le site consommateur.

L'***InterDock*** est une organisation intermédiaire entre le producteur et les sites utilisateurs. Une entreprise peut comprendre plusieurs *InterDock*. Comme ils fournissent des informations globales, ils permettent de créer une vue globale de l'entreprise.

3.1.2. Description de l'information

Les fichiers DSD (*Deployable Software Description*) [HHW99-b, HHW98], spécifiés en XML [Xml], permettent de modéliser les applications et les différents sites.

Le serveur est décrit par le fichier *releasedock.dsd*, qui spécifie la localisation du serveur et les applications qu'il peut déployer. Pour chacune de ces applications, le fichier contient la localisation du fichier DSD la décrivant et le nom de la famille à laquelle elle appartient.

Le client est décrit par le fichier *fielddock.dsd*, qui indique la localisation du client et les informations matérielles (telle que l'architecture) et logicielles (comme le système d'exploitation et les applications déjà présentes sur le site).

Chaque application est aussi décrite par un fichier DSD, contenant une série de propriétés. Une propriété a plusieurs attributs : un nom, un type (chaîne de caractères, entier ou booléen) et une valeur par défaut. Les descripteurs d'applications sont basés sur une extension de celui d'OSD [Osd]. Pour une propriété donnée, il peut aussi y avoir un ensemble de valeurs possibles (pour certains types seulement). Il y a deux types de propriétés :

- Les **propriétés externes** permettent de décrire le site utilisateur (propriétés sur le système d'exploitation, par exemple).
- Les **propriétés internes** décrivent les caractéristiques du système lui-même (la version du logiciel, par exemple).

Les propriétés sont des informations qui sont utilisées pour choisir l'application à déployer (celle qui correspond le mieux aux caractéristiques de la cible). Des contraintes sont aussi utilisées pour déterminer ce qui est nécessaire pour que l'application puisse fonctionner correctement. Le fichier DSD contient donc aussi les contraintes [HHW99-d], caractérisant les exigences matérielles ou logicielles nécessaires pour la bonne exécution du logiciel installé. Il y en a deux catégories :

- Les **assertions** sont des contraintes qui doivent être vérifiées mais qui ne peuvent pas être résolues en cours de déploiement. Par exemple, une contrainte peut imposer le système d'exploitation *Windows*. Si cette contrainte n'est pas vérifiée, le déploiement échouera.
- Les **dépendances** peuvent être résolues en cours de déploiement. Par exemple, si la présence d'un autre logiciel est nécessaire, celui-ci peut être installé, et le déploiement pourra se poursuivre dans de bonnes conditions.

En outre, les artefacts, fichiers physiques composant le logiciel, sont regroupés en collection. Pour chacun d'eux, son nom et sa localisation (avant et après le déploiement) sont spécifiés. Pour chaque fichier DSD, on définit aussi les agents qui seront fournis par le système (agent d'installation, de mise à jour, etc...).

3.1.3. Les activités supportées

Plusieurs activités du cycle de vie sont mises en œuvre par Software Dock [HHC+98] : l'installation, la mise à jour, la reconfiguration, la vérification de cohérence et la désinstallation.

3.1.3.1. L'installation

Tout d'abord, l'agent d'installation est téléchargé vers le *FieldDock* (utilisateur). Cette opération peut se faire à l'initiative de l'utilisateur, ou être ordonnée par un autre agent, dans le cas de dépendance d'un logiciel par rapport à un autre. Cet agent récupère le registre de configuration du site et les dépendances du logiciel à installer. S'il y a des dépendances et que l'un des logiciels n'est pas installé, l'agent demande au serveur de télécharger les agents d'installation concernés (un pour chaque logiciel à installer). Après avoir résolu toutes les dépendances, l'agent récupère la bonne configuration de l'application. Puis, il crée un nœud application pour le logiciel dans le registre du *FieldDock* ; un événement est généré et reçu par l'agent du site de l'utilisateur. Celui-ci crée un répertoire pour l'application. Ensuite, l'agent d'installation insère les nœuds des fichiers et répertoires de l'application. Chacune de ces insertions génère un événement et l'agent de site crée ainsi toute l'arborescence physique de l'application. En fin d'installation, l'agent de mise à jour est ajouté au site du *FieldDock*.

3.1.3.2. La mise à jour

Dès qu'une nouvelle version d'une application est disponible, elle est ajoutée au registre du *ReleaseDock* (producteur). Cette insertion génère un événement qui est reçu par l'agent de mise à jour du *FieldDock* (s'il avait souscrit à ce type d'événements). Cet agent recherche le registre du *FieldDock*. Puis, il communique avec le *ReleaseDock* pour retrouver la bonne configuration du logiciel. Enfin, il insère les nœuds des fichiers et répertoires correspondants, et l'agent de site modifie l'arborescence en fonction.

3.1.3.3. La reconfiguration et la vérification de cohérence

L'activité de reconfiguration consiste à réinstaller le logiciel sous une nouvelle configuration. Par exemple, la reconfiguration peut permettre d'installer une version qui utilise moins d'espace disque sur une machine qui commence à être surchargée.

Vérifier la cohérence d'un logiciel, c'est vérifier que tous les fichiers et répertoires de l'application sont présents.

3.1.3.4. La désinstallation

Il s'agit de supprimer (de manière physique) une application installée, et d'enlever les nœuds des fichiers et répertoires de l'application.

3.1.4. Conclusion

Même si l'outil Software Dock ne permet pas de personnaliser entièrement le déploiement d'un logiciel (pas de choix au niveau des outils utilisés, pas de stratégies de déploiement), il a quelques points forts. En effet, il offre la possibilité de choisir une configuration particulière du logiciel. De plus, il permet d'exprimer un grand nombre de contraintes matérielles (liste extensible), et de gérer les contraintes logicielles (dépendances) avec le modèle d'application. Il offre aussi la possibilité de reconfigurer un logiciel ou de vérifier sa cohérence.

Parmi les points faibles, on peut citer qu'il n'y a pas de notification et que la planification d'une installation n'est pas possible. De plus, un package est toujours destiné à un seul utilisateur : on aurait apprécié de pouvoir utiliser la notion de groupe de machines.

3.2. CADeComp

Depuis quelques années, l'INT d'Evry [Int] s'intéresse au déploiement automatique d'applications multi-composants [ATB03, ATB04]. Leur approche, *CADeComp*, s'intéresse particulièrement au déploiement d'applications sensibles au contexte. Pour l'étudier, nous présentons d'abord le contexte de travail. Nous définissons ensuite la sensibilité du déploiement au contexte. Enfin, nous présentons l'architecture et le modèle de données utilisés.

3.2.1. Contexte scientifique

Le travail réalisé par l'INT est placé dans un contexte où les sites clients sont mobiles et connectés au réseau [TPB03]. Le procédé de déploiement est alors déclenché lors d'un accès à un service. Quand l'utilisateur quitte ce service, la désactivation et la désinstallation du (des) composant(s) concerné(s) est déclenchée. Ainsi, l'interface utilisateur est le seul composant toujours nécessaire sur le terminal client. De plus, dans ce cas, l'activité de mise à jour n'est plus nécessaire, puisque le composant est ré-installé à chaque utilisation.

Le déploiement en temps réel permet aussi de personnaliser les services déployés, dans un contexte d'exécution précis. Ainsi, chaque client a un composant, réalisant le service demandé, correspondant le plus aux attentes et aux caractéristiques de la cible. Pour cela, le contexte d'exécution est collecté avant le déploiement. Ce contexte inclut notamment les capacités du terminal en termes matériels, les interfaces utilisateur, l'autonomie de batterie, la capacité CPU et les logiciels déjà disponibles. Le contexte comprend aussi la position géographique de l'utilisateur et ses préférences. Avec l'ensemble de ces informations, le déploiement est ensuite adapté aux ressources disponibles.

De plus, la division de l'application en plusieurs composants permet, si cela est nécessaire, l'exécution de chaque composant sur des sites différents. Le procédé de déploiement, quant à lui, prend en compte les propriétés statiques, mais aussi les propriétés dynamiques du site, au moment du déploiement.

3.2.2. Sensibilité du déploiement au contexte

L'un des principaux objectifs de CADeComp est de supporter la sensibilité du déploiement au contexte [ATS05]. Le contexte est défini comme toute information caractérisant la situation d'une entité (personne, localisation, objet, …) qui peut être considérée comme pertinente pour

l'interaction de l'utilisateur avec son application. Ainsi, une application sensible au contexte est définie comme une application capable de percevoir et d'analyser son contexte et de s'y adapter.

Plusieurs types d'information de contexte peuvent affecter le déploiement, comme, par exemple :

- les informations relatives aux **environnements matériels et logiciels** des sites qui vont accueillir l'application à déployer,
- les informations relatives à l'**utilisateur final** comme ses préférences, son identité et sa localisation,
- plusieurs autres types de contextes qui sont liés à la **sémantique de l'application** (par exemple, la température peut avoir un impact sur le déploiement d'une application d'informations touristiques).

De plus, le déploiement d'applications à base de composants nécessite la description de plusieurs paramètres qui peuvent varier en fonction du contexte, comme :

- l'**architecture de l'application** : les composants de l'application et leurs connexions peuvent changer selon le contexte.
- le **choix des versions d'implémentation des composants** : la version d'un composant à déployer peut être différente selon le contexte.
- la **sélection de l'emplacement des composants** : chaque composant de l'application peut être déployé sur un terminal différent, en fonction de la disponibilité des ressources et de la localisation de l'utilisateur.
- les **valeurs des propriétés de configuration** : les propriétés fonctionnelles (contrôlant le comportement du composant) et non fonctionnelles (persistance, politiques de sécurité, …) du composant peuvent différer suivant le contexte. Par exemple, la propriété *langage de l'utilisateur* peut avoir des valeurs différentes selon le contexte de déploiement.

CADeComp est une plate-forme qui permet de prendre en compte les informations du contexte de déploiement pour faire varier ces différents paramètres.

3.2.3. Architecture et modèle de données

Un méta-modèle de contexte a été défini pour les services sensibles au contexte, et appelé COMCAS (*COntext Meta-model for Context-Aware Services*) [ABT+05]. Comme le montre la Figure 3.6, COMCAS se divise en quatre domaines : contexte, capteur, règles et politiques, et services sensibles au contexte (*ServicesSC*).

Le **domaine de contexte** représente les informations de contexte auxquelles les services sont sensibles. Les informations de contexte directes sont récupérées par des capteurs, tandis que les informations indirectes sont déduites d'autres contextes en utilisant des règles d'interprétation.

Le **domaine de capteur** permet la représentation des capteurs qui interagissent.

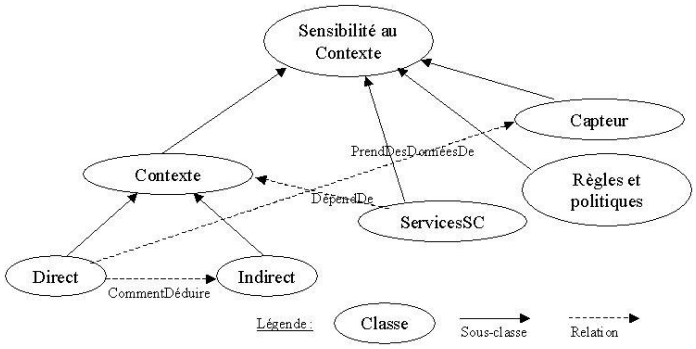

Figure 3.6 Représentation de COMCAS

Le **domaine de règles et politiques** représente les politiques d'adaptation et les méthodes d'interprétation qui sont utilisées pour s'adapter au contexte.

Enfin, le **domaine de services sensibles au contexte** représente les services installés au dessus du *middleware* sensible au contexte proposé.

En outre, l'adaptation d'une application à son contexte d'utilisation nécessite trois étapes :

- la collecte des informations de contexte,
- l'analyse des informations de contexte et la décision des activités d'adaptation,
- l'exécution des activités d'adaptation.

Ces étapes à réaliser permettent de définir l'architecture du service de déploiement, proposé et placé au dessus d'une infrastructure sensible au contexte, qui communique avec les sources d'information du contexte. La Figure 3.7 illustre cette architecture [ATS+05].

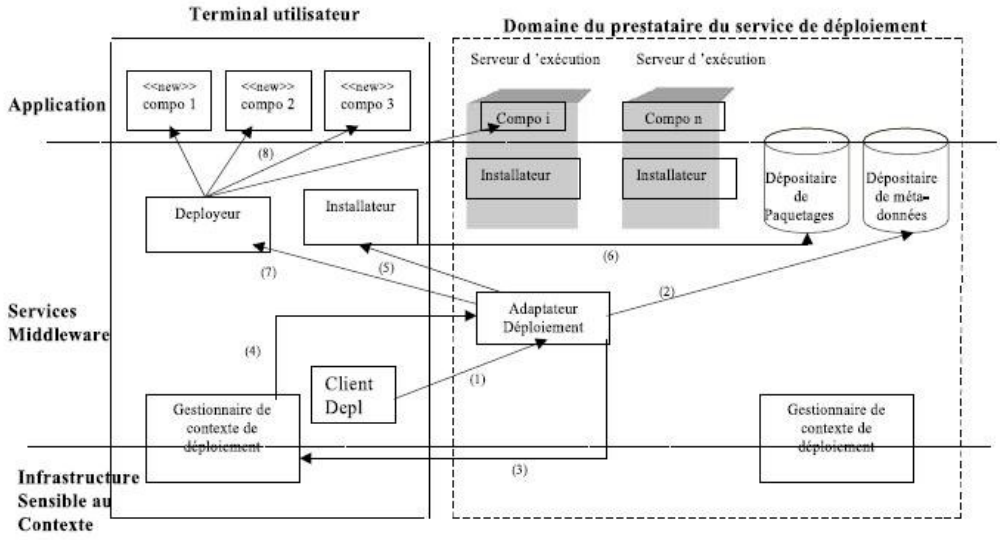

Figure 3.7 Architecture et étapes de déploiement de la plate-forme de déploiement sensible au contexte

Les **dépositaires** (de paquetages et de méta-données) contiennent les composants et les descripteurs nécessaires au déploiement des applications et à leur adaptation au contexte. Le composant **Installateur** permet l'installation locale des composants sur un terminal utilisateur.

Le **gestionnaire de contexte** joue le rôle de passerelle entre le service de déploiement et l'infrastructure sensible au contexte. Il envoie ces informations à l'adaptateur de déploiement. L'**adaptateur de déploiement** analyse les contextes pertinents et détermine l'assemblage de l'application à déployer en utilisant les méta-données nécessaires. Il applique ensuite des règles d'adaptation (correspondant aux contextes vérifiés) et produit un descripteur de déploiement. Enfin, le « **déployeur** » se charge de déployer l'application en utilisant le descripteur d'assemblage produit.

Un algorithme de placement des composants sur les nœuds a aussi été défini [BTA+05]. La première partie de cet algorithme concerne la sélection et permet de choisir quelle implémentation sera déployée et quels nœuds du réseau seront utilisés. La deuxième partie de l'algorithme consiste à affecter les implémentations sélectionnées aux nœuds choisis.

3.2.4. Conclusion

CADeComp permet le déploiement d'applications multi-composants. Le point fort est que le déploiement des composants est réalisé en fonction du contexte de déploiement. Cependant, cette approche est très ciblée pour les clients mobiles, puisque les composants sont installés à chaque nouvelle demande de service, et désinstallés à chaque arrêt du service (pas de mise à jour).

En outre, le service de déploiement n'a pas de capacité de raisonnement automatique : il repose entièrement sur les règles spécifiques d'adaptation associées à l'application. D'autres règles pourraient aussi être utilisées pour imposer des contraintes sur le réseau, comme, par exemple, ne pas déployer sur un site déjà surchargé.

3.3. Resolvit

Resolvit est le prototype de l'approche proposée par l'Université Polytechnique de Madrid [Upm]. Resolvit est un moteur de déploiement pour OSGi [Osg03]. L'approche souhaite aussi couvrir l'ensemble des activités du cycle de vie du déploiement. Pour l'illustrer, nous présentons tout d'abord le contexte de travail. Puis, nous nous focalisons sur la gestion des dépendances, point intéressant dans l'approche, en déterminant les besoins et la manière utilisée pour les résoudre [RDU04].

3.3.1. Contexte scientifique

Resolvit s'intéresse particulièrement au déploiement d'architectures orientées service (SOA) [BC01, Bou03]. Il s'agit de déployer, de manière distribuée, des unités fournissant des services, sur un ensemble de plates-formes avec une configuration spécifique. De manière générique un service est une pièce de logiciel qui offre une certaine fonctionnalité. Plusieurs implémentations du même service peuvent donc exister.

Les cibles de déploiement sont connectées à Internet et un ensemble de serveurs fournissent des unités de déploiement. De plus, un centre de gestion est en charge des activités de

déploiement. Il doit résoudre les problèmes de dépendances et de conflits entre les unités à déployer.

L'approche est orientée sur le déploiement de services OSGi. Un service OSGi est défini par une interface Java et implémenté par un objet. Pour être utilisés par d'autres, les services doivent être enregistrés dans le *service registry*. Au moment de l'enregistrement, une implémentation d'un service peut spécifier des informations qui permettent de la différencier des autres implémentations. Ainsi, lors du déploiement, une sélection de l'implémentation souhaitée peut être réalisée.

Dans OSGi, l'unité de déploiement est le *bundle*. Un bundle est un fichier compressé (*zip*) qui contient des classes Java, un activateur de bundle, un fichier de méta-données et d'autres ressources. De plus, un bundle a un cycle de vie qui lui permet de passer tour à tour par l'un des états suivants : installé (*installed*), résolu (*resolved*), démarré (*started*), en cours de démarrage (*starting*), en cours d'arrêt (*stopping*), arrêté (*stopped*), désinstallé (*uninstalled*).

La partie suivante s'intéresse aux exigences souhaitées du moteur de déploiement qui a pour but de résoudre automatiquement les dépendances et conflits entre unités de déploiement.

3.3.2. Expression des besoins

En premier lieu, l'état de la plate-forme doit pouvoir être stocké. En effet, il faut pouvoir consulter l'état de la plate-forme pour adapter le déploiement, en fonction des caractéristiques de la cible. De plus, les tâches de déploiement doivent être fiables : soit le déploiement se termine correctement, soit la plate-forme est laissée dans son état initial. Dans le cas où le déploiement se déroule correctement, les changements de l'état de la cible doivent être persistants. Pour cela, Resolvit décompose le déploiement en deux phases : vérification et exécution. Si les vérifications sont correctes, alors le déploiement est exécuté automatiquement.

De plus, les dépendances doivent être spécifiées dans des descripteurs de déploiement afin de pouvoir gérer leur résolution. Pour OSGi, la spécification des dépendances se fait au niveau service. Les versions des différentes ressources doivent aussi être mentionnées pour les distinguer les unes des autres.

Pour automatiser et adapter le déploiement à la plate-forme cible, le moteur de déploiement doit aussi permettre l'accès et l'analyse des données stockées dans les descripteurs de déploiement (spécification des dépendances, contexte de déploiement, …). Pour cela, Resolvit propose un accès local ou distant aux informations.

Enfin, le moteur de déploiement doit aussi prendre en compte les conditions de sécurité et les coûts du service. En effet, le coût du déploiement doit rester raisonnable, et, les conditions de sécurité doivent être respectées, afin de ne pas déployer des unités non désirées.

3.3.3. Méta-données et algorithme de déploiement

Un format de descripteur de déploiement a été créé pour Resolvit, utilisant un schéma XML [XmlS]. La gestion des dépendances est particulièrement importante dans le déploiement des architectures orientées service, c'est pourquoi les dépendances sont explicitement et systématiquement décrites dans le descripteur.

Le descripteur de déploiement contient les informations suivantes :

- information descriptive : nom, services offerts, version, fournisseur, … ;
- dépendances avec d'autres entités (services, dans le cas d'OSGi) : type des dépendances, services requis, …;
- information sur les ressources nécessaires : espace disque, processeur, système d'exploitation, … .

Le procédé de résolution des dépendances dépend du type de dépendance. Plusieurs types de dépendances sont définis :

- ***AND*** : la dépendance est obligatoire. Cela signifie qu'une entité en a besoin d'une autre pour s'exécuter.
- ***NOT*** : il existe un conflit. Dans ce cas, deux entités ne peuvent pas cohabiter sur une même cible.
- ***OR*** : une ou plusieurs entités peuvent être sélectionnées en même temps.
- ***XOR*** : deux entités peuvent être sélectionnées, mais pas en même temps.

A partir de ces informations, un algorithme de déploiement peut être défini. L'algorithme est divisé en deux phases : résolution des dépendances et exécution. La phase de résolution des dépendances est décrite par la Figure 3.8.

Pour résoudre les dépendances, l'algorithme commence par lire le descripteur de déploiement de l'unité de déploiement. Ensuite, l'état de la plate-forme est analysé. Si l'entité (service, dans le cas d'OSGi) requise est déjà installée, le procédé de déploiement est terminé. Si l'entité n'est pas installée ou si ce n'est pas la dernière version qui est installée, les dépendances sont lues. S'il y a des dépendances, elles sont ajoutées à la structure de données qui permet de stocker l'information collectée. La phase suivante consiste, ensuite, à résoudre les conflits et installer les ressources.

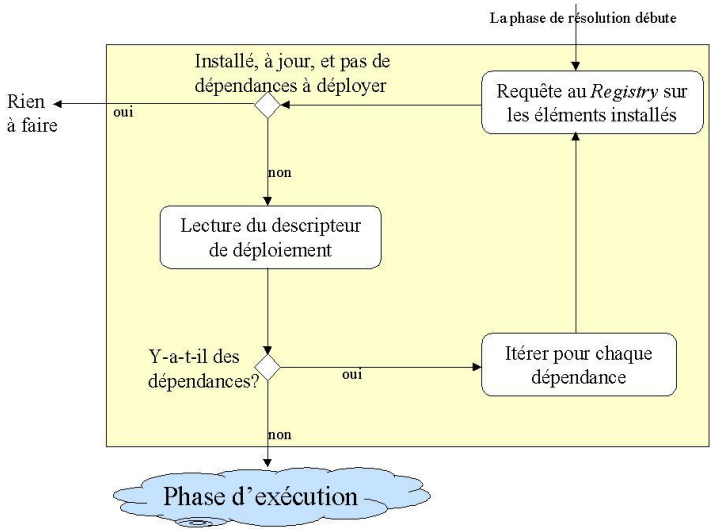

Figure 3.8 Phase de résolution des dépendances de l'algorithme de déploiement de Resolvit

3.3.4. Conclusion

L'approche de l'UPM est basée sur le déploiement des architectures orientées services. L'implémentation de Resolvit a été expérimentée avec OSGi. L'un des points forts de l'approche est la résolution de dépendances. Il est en effet intéressant de pouvoir classifier les différents types de dépendances et d'adapter leur résolution en fonction de ces types. De plus, pour résoudre ces dépendances automatiquement, un algorithme a été défini.

Un autre point positif est la prise en compte du contexte de déploiement, pour adapter le déploiement au plus proche des caractéristiques des cibles. De plus, la cohérence de l'état de la plate-forme cible est assurée avec le déploiement en deux phases : une unité n'est installée que si elle peut fonctionner.

Cependant, le traitement des erreurs au cours de la phase d'exécution du procédé de déploiement n'est pas traité. En effet, même si toutes les dépendances ont pu être résolues et que le procédé de déploiement est lancé, il peut, par exemple, se produire une erreur au cours du transfert des ressources.

En outre, l'approche est spécialisée pour les architectures orientées service et ne permet donc pas le déploiement d'applications plus générales. Enfin, un niveau permettant à une entreprise d'appliquer des choix au cours du déploiement est absent. Par exemple, il n'est pas possible d'exprimer le souhait de déployer la même version d'une unité (même implémentation d'un service) pour un ensemble de cibles.

3.4. Discussion et synthèse sur les environnements de déploiement issus de la recherche

Les environnement de déploiement que nous avons présentés dans cette partie se focalisent tous sur des problématiques différentes. Nous les avons choisis pour avoir un large aperçu de ce qui peut-être utile pour le déploiement d'applications à grande échelle.

Tout d'abord, Software Dock fournit la base d'un support automatisé pour le déploiement. En effet, l'utilisation de modèles permet de réaliser les différentes activités du déploiement de manière automatique. De plus, l'utilisation de contraintes permet de s'assurer qu'une application déployée peut fonctionner correctement sur un site cible donné.

Ensuite, CADeComp, même s'il s'agit d'une approche ciblée pour les clients mobiles, permet de prendre en compte le contexte de déploiement. Ainsi, les applications (ou composants d'applications) peuvent imposer un certain nombre de contraintes sur les clients qui les reçoivent. Cependant, un niveau entreprise manque pour ajouter la définition de contraintes au niveau de l'entreprise ou des sites clients (comme par exemple, de ne pas déployer de nouveaux composants sur un site déjà surchargé).

Enfin, Resolvit est une approche particulièrement intéressante pour sa classification des dépendances et leur traitement. De plus, le contexte de déploiement est pris en compte et la cohérence des sites cibles est assurée. En effet, seules des applications pouvant fonctionner correctement peuvent être déployées. L'ajout du traitement des erreurs au cours de l'exécution du procédé de déploiement serait un atout supplémentaire pour l'approche.

Notre approche doit prendre en compte ces différents avantages. En effet, il faut pouvoir :

- fournir un **support automatisé**, pour toutes les activités du cycle de vie du déploiement (comme Software Dock),
- prendre en compte le **contexte de déploiement** (comme CADeComp), pour ne déployer que des applications qui fonctionnent correctement et assurer la cohérence des sites cibles,
- prendre en compte les **différents types de dépendances**, pour pouvoir les traiter différemment (comme Resolvit).

De plus, il faut aussi traiter une problématique qui est absente de ces trois approches : l'administration du déploiement au niveau entreprise. Ce niveau permet d'ajouter des stratégies de déploiement et de personnaliser le déploiement. Les trois approches que nous avons présentées prennent en compte des contraintes au niveau des unités de déploiement (applications et/ou composants), mais ne considèrent pas les contraintes qui peuvent être ajoutées par l'entreprise cible de déploiement. La gestion de stratégies de déploiement permet alors de prendre en compte de telles contraintes.

4. DES OUTILS COMMERCIAUX POUR LE DEPLOIEMENT

Plusieurs outils commerciaux de déploiement sont disponibles. Chacun d'eux fournit des services variés et propose un niveau d'abstraction différent. Dans cette partie, nous présentons brièvement deux outils dont l'objectif principal est le déploiement à grande échelle dans une entreprise : Tivoli et Rembo Auto-Deploy. Nous terminons ensuite cette partie en discutant autour d'autres outils du commerce ayant des objectifs plus ou moins ciblés.

4.1. Tivoli

Tivoli [Tivoli] est un environnement de déploiement et d'administration proposé par IBM [Ibm], pour gérer des réseaux et applications. Cet environnement propose plusieurs outils réalisant chacun des tâches particulières. Nous présentons ici trois d'entre eux, utiles pour le déploiement d'applications.

4.1.1. Le gestionnaire de configuration

Le gestionnaire de Configuration (*Tivoli Configuration Manager* [Tcm]) est une solution pour déployer des logiciels et assurer le suivi des configurations matérielles et logicielles des machines d'une entreprise. Il propose un contrôle total des logiciels et matériels de l'entreprise. Le déploiement d'applications complexes peut être réalisé sur plusieurs sites, à partir d'un point central du réseau.

Cet outil permet de distribuer et de réaliser l'inventaire des logiciels sur les machines ciblées par le « déployeur ». Ces étapes sont effectuées grâce à la prise en charge d'annuaires, grâce au module d'inventaire et à l'outil de distribution. De plus, ces opérations peuvent être sécurisées à travers des pare-feux.

Le gestionnaire de configuration de Tivoli regroupe une distribution de logiciels et un inventaire logiciel et matériel, dans un environnement multi plate-formes. Le module d'inventaire permet de déterminer les ressources de l'entreprise, tandis que l'outil de

distribution permet de configurer et d'installer des applications, de manière centralisée et automatisée.

La distribution des logiciels peut aussi être planifiée. En outre, les modifications apportées au système et à la configuration des machines peuvent être surveillées pour ajouter des fonctionnalités de rapport.

4.1.1.1. L'inventaire

Un module d'inventaire permet de réaliser l'inventaire matériel et logiciel de chaque machine de l'entreprise. Ceci implique que ce module doit tout d'abord collecter les informations nécessaires, puis mettre à jour la base contenant l'information liée à toute l'entreprise.

La recherche d'information se fait en introspectant les machines. Puis les informations collectées (type, localisation, …) sont centralisées pour avoir une vision globale des diverses configurations de l'entreprise.

L'inventaire peut être utilisé, à diverses étapes du déploiement, pour :

- vérifier les contraintes matérielles et logicielles lors du déploiement d'une application,
- déterminer les logiciels déployés dans l'entreprise (ces données peuvent aussi être utilisées pour planifier le déploiement),
- vérifier la configuration logicielle d'une machine (vérifier la cohérence des données).

4.1.1.2. La distribution

L'outil de distribution (*Tivoli Software Distribution* [Tsd]) supporte l'installation, la configuration et la mise à jour des logiciels, à distance et de manière centralisée. Pour préparer le déploiement, cet outil permet notamment de définir les dépendances logicielles et de choisir les actions à exécuter pour réaliser le déploiement.

Pendant le déploiement, il est aussi possible de configurer l'application et de vérifier les versions des composants dépendants. L'outil permet aussi d'exécuter les actions spécifiées précédemment.

4.1.2. La console d'entreprise

La console d'entreprise (*Tivoli Enterprise Console* [Tec]) fournit des fonctions d'identification des problèmes qui peuvent survenir dans le déploiement ou l'administration du logiciel et du matériel de l'entreprise. Cet outil prend en charge un certain nombre d'applications et de périphériques.

Il permet aussi d'analyser les événements et les problèmes qui surviennent. Ces événements peuvent être traités de manière automatique ce qui améliore la gestion automatique du déploiement et accélère cette étape.

4.1.3. Le gestionnaire de licence

Le gestionnaire de licence (*Tivoli License Manager* [Tlm]) est un outil de gestion des licences des différents logiciels de l'entreprise. Cet outil fournit un inventaire des licences logicielles :

- détenues par l'entreprise,
- utilisées par l'entreprise,
- dont l'entreprise pourrait avoir besoin.

Il comporte aussi un agent qui permet de mettre à jour les logiciels automatiquement dès qu'une nouvelle version est disponible.

Un outil de gestion des licences est utile pour les entreprises. Il permet de déterminer quelles licences sont nécessaires à l'entreprise, et ainsi, de supprimer celles qui sont inutiles. Il permet aussi de contrôler les licences à acheter et de planifier l'évolution du parc informatique de l'entreprise.

4.1.4. Conclusion

Tivoli propose une solution assez complète de gestion des configurations matérielles et logicielles d'une entreprise. Le point fort de l'approche est la vision entreprise du déploiement, avec des outils qui répondent aux besoins des organisations gérant un grand nombre de machines. Le déploiement des applications peut être planifié. Un inventaire des configurations (matérielles et logicielles) des machines permet de les gérer plus facilement et de proposer aussi un outil pour traiter les licences logicielles.

De plus, le déploiement et l'administration peuvent se faire à distance mais uniquement de manière centralisée. Par ailleurs, le procédé de déploiement est fixe, ce qui limite la flexibilité pour l'utilisateur.

4.2. Rembo Auto-Deploy

Rembo Auto-Deploy [Rad] (de Rembo Technology [Rembo]) est un outil créé pour la gestionet le déploiement à distance d'un grand nombre de machines. Dans un environnement réseau, il permet de configurer des machines, de déployer des applications et de restaurer une configuration sur une machine. Pour présenter cet outil, nous commençons par énoncer les possibilités qu'il offre, puis nous définissons quelques concepts qu'il utilise.

4.2.1. Possibilités offertes par l'outil

Rembo Auto-Deploy (RAD) permet de déployer des images logicielles, de les cloner, de réaliser des installations automatisées et de personnaliser des machines. Des mises à jour ou des récupérations de configuration peuvent aussi être programmées. L'ensemble des opérations peut être lancé et contrôlé à distance avec la console de gestion proposée par l'outil.

4.2.1.1. Techniques de clonage

Plus en détails, RAD assemble des configurations d'ordinateurs cibles à partir de blocs de construction variés. Le premier bloc comporte l'image du système d'exploitation. Ce bloc est

une copie du contenu du disque d'un ordinateur de référence. Cette copie est préparée en utilisant l'outil de préparation de système de Microsoft : SYSPREP [SysP].

Les autres blocs de construction sont des packages logiciels. L'image du système d'exploitation et les images des applications sont ensuite combinées et écrites sur le disque dur en utilisant les techniques de clonage de Rembo [RCl].

4.2.1.2. Déploiement à grande échelle

Différentes configurations de machine peuvent être déployées simultanément sur un ensemble de machines. Un protocole de transport *multicast* crypté est alors utilisé. Le *multicasting* permet d'optimiser l'utilisation du réseau, puisque les fichiers nécessaires à plusieurs machines sont envoyés une seule fois, au lieu d'être envoyés individuellement.

De plus, pour réduire l'espace disque nécessaire pour stocker les diverses images de disque utilisées, des copies multiples du même fichier ne sont conservées qu'une seule fois dans le dépôt. L'assemblage des différentes images permet ensuite de réaliser les diverses configurations utilisées.

4.2.1.3. Console de gestion centrale

Le procédé de déploiement est conduit par une base de données centrale. Celle-ci contient les paramètres uniques de chaque machine. Ces paramètres incluent des règles qui permettent de décider quelles images et quels logiciels déployer sur chaque machine.

Le procédé de déploiement peut être entièrement automatisé, sans interaction sur la machine cliente. Cependant, RAD peut aussi laisser le choix à l'utilisateur de déterminer quel système d'exploitation utiliser (*dual boot*) et quelles applications déployer. Des rapports sont aussi disponibles pour contrôler le déploiement depuis la console centrale.

4.2.1.4. Maintenance des images

RAD simplifie la maintenance des images en séparant l'image de base des applications. Les packages d'application sont créés sur une plate-forme spécifique (machine de référence) et peuvent être déployés facilement sur d'autres machines.

RAD offre aussi la possibilité de faire des images (*snapshot*) du disque. Ainsi, il est possible de comparer les images du disque avant et après l'installation d'une application et de déterminer quels changements ont été effectués.

De plus, RAD permet de faire du re-déploiement. Cette technique peut notamment être intéressante pour les parcs informatiques où les machines sont utilisées par beaucoup d'utilisateurs et où chaque utilisateur n'a pas de machine personnelle (comme les universités par exemple). En effet, il est possible de restaurer l'image précédente du disque après la déconnexion d'un utilisateur. Ainsi, toutes les modifications réalisées par celui-ci sont effacées. Pour cela, il faut conserver l'image principale du disque sur une partition non partagée du disque local. Ensuite, RAD restaure la machine en défaisant les modifications apportées.

4.2.2. Concepts

Pour réaliser les différentes étapes citées précédemment, plusieurs concepts sont utilisés par RAD. Nous présentons ici le procédé de déploiement et les éléments nécessaires à son exécution par l'outil. Puis, nous énonçons quelques scénarios.

4.2.2.1. Procédé de déploiement

Pour RAD, un déploiement est un procédé d'installation d'un système d'exploitation sur une machine. Le déploiement inclut aussi la configuration du système pour des utilisateurs spécifiques.

Plusieurs étapes sont exécutées séquentiellement pour réaliser le procédé d'installation :

- Les partitions du disque sont créées (et formatées), et les packages logiciels sont téléchargés dans un endroit de stockage temporaire.
- Les fichiers du système d'exploitation sont copiés sur les partitions du disque concernées.
- La configuration spécifique de l'hôte (comme le nom de l'hôte ou la clé du produit) est récupérée dans la base de données centrale pour créer le fichier de configuration utilisé par l'outil SYSPREP.
- Le système d'exploitation est lancé et SYSPREP le configure avec les informations stockées dans la base de données de RAD. Quand ceci est terminé, RAD reprend la main pour afficher un message indiquant la fin du déploiement.

Quand le déploiement est terminé, le système d'exploitation est prêt à être utilisé.

4.2.2.2. Eléments nécessaires pour réaliser un déploiement

Pour commencer un déploiement sur une machine cliente, RAD a besoin de divers éléments :

- Un **schéma de déploiement** est associé à la machine cible. Ce schéma est utilisé pour déterminer comment déployer le système d'exploitation sur la cible (par exemple, en utilisant un transfert de fichiers *unicast* ou *multicast*). S'il n'y a pas de schéma de déploiement associé à la cible, un schéma « par défaut » est disponible dans RAD.
- Une **configuration du système d'exploitation** est utilisée pour sélectionner quel système installer. Si aucune configuration n'est associée à la cible, le déploiement ne peut pas démarrer.
- Des **packages logiciels** peuvent être ajoutés pour être déployés, en plus du système d'exploitation, pendant le procédé de déploiement.

La configuration et les packages logiciels permettent de déterminer quoi déployer sur la cible. Le schéma de déploiement, quant à lui, détermine comment déployer.

La base de données de RAD conserve ensuite les associations entres les cibles et les schémas de déploiement, entre les cibles et les configurations, et, entre les cibles et les packages logiciels. Ces associations peuvent être réalisées manuellement ou à l'aide de règles. Des

règles peuvent, par exemple, exprimer de déployer *WindowsXP* sur les machines dont le nom commence par *Dell*. Conserver l'ensemble de ces associations permet de savoir à tout moment ce qui est déployé sur chaque machine.

4.2.2.3. Les scénarios possibles

Selon le nombre de configurations associées à une machine cible, la machine peut se comporter différemment lorsqu'elle démarre sur le réseau :

- Si aucune configuration n'est disponible, un écran est affiché (sur la console centrale) pour demander à l'utilisateur de définir une configuration à associer à la cible.
- Si une ou plusieurs configurations sont associées à la cible, mais qu'aucun déploiement n'a été programmé, un écran affiche la liste des configurations possibles. Le déploiement débute après avoir choisi (interactivament) l'un des éléments de la liste,
- Si une ou plusieurs configurations sont associées à la cible, et un déploiement a été programmé pour une configuration spécifique, la cible commence son déploiement sans intervention de l'utilisateur.

4.2.3. Conclusion

RAD permet de gérer un ensemble de machines et peut ainsi facilement être utilisé par une entreprise. Cette gestion s'effectue de manière centralisée, à distance depuis un serveur.

Cet outil utilise des images de disque pour déployer des applications. Ceci limite le déploiement, puisque chaque application doit être déployée au moins une fois sur une machine de référence pour pouvoir réaliser l'image du disque. Cependant, un avantage de cette technique est qu'elle permet de partir d'une machine complètement « nue » et de déployer le système d'exploitation à partir de son image. Cette méthode est malheureusement réservée au système d'exploitation *Windows*, puisqu'elle utilise l'un de ses outils.

De plus, RAD permet le re-déploiement. Cette activité n'a pas été mentionnée précédemment dans le cycle de vie du déploiement (cf. paragraphe 4 du Chapitre 2). Cependant, elle est particulièrement intéressante pour les organisations qui gèrent un grand nombre de machines sans utilisateur attitré. Les machines peuvent ainsi aisément retrouver une image de base, compatible avec le profil de chaque utilisateur.

Enfin, l'utilisation de règles pour associer des configurations spécifiques à des machines particulières est intéressante et peut s'apparenter à des stratégies de déploiement. En effet, cette méthode permet d'associer un type de machine (ou éventuellement, un type d'utilisateur) à un type de configuration. Cependant, aucun moyen n'est disponible pour grouper les machines plus hiérarchiquement, par des moyens autres que leurs caractéristiques (par exemple, groupe des machines, à caractéristiques différentes, du service de comptabilité). De plus, l'utilisation de règles ne permet pas d'avoir une vision globale des groupes de machines ainsi formés.

De plus, d'autres stratégies pourraient être définies. Par exemple, si l'entreprise ne souhaite déployer que des applications implémentées en java sur un type de machine, elle doit vérifier « manuellement » que la configuration associée au groupe (via une règle) ne contient que ce

type d'application (packages logiciels). Il pourrait être intéressant de définir de nouveaux outils pour élargir les stratégies possibles.

4.3. Autres outils de déploiement

Les technologies actuellement utilisées dans l'industrie visent chacune différents objectifs ou utilisent diverses approches. Elles fournissent alors des services plus ou moins complexes. Carzaniga et al. [CFH+98] ont classé ces technologies en différentes catégories : installateurs, gestionnaires de package, systèmes de gestion d'application, ... Ici, nous décrivons brièvement quelques caractéristiques, en relation avec le déploiement, de quelques uns de ces outils, choisis parmi les plus connus.

4.3.1. Les installateurs et l'outil InstallShield

Les installateurs packagent un système logiciel dans une archive capable de s'auto-installer. Ensuite, l'archive est distribuée sur les machines clientes, par le réseau ou par Internet. La plupart de ces outils sont capables de gérer aussi la désinstallation en défaisant les étapes réalisées pendant l'installation.

Le plus connu de ces outils est *InstallShield* [IS11]. Il est réservé aux applications *Windows* [Win], puisqu'il utilise l'un des services du système d'exploitation : le service *Windows Installer* [WinI]. Ce service permet d'installer ou de désinstaller des applications de manière automatisée. Il permet aussi de réparer des fichiers corrompus et offre un service de reprise pour revenir à l'état initial en cas de problème en cours d'installation.

InstallShield permet de créer un package qui est ensuite distribué aux clients et exécuté. Avec un package, des informations sont disponibles pour le décrire (nom, date de création, ...), les instructions d'installation (fichiers à copier, registres à créer, ...), les références vers les ressources de l'application. Après sa création, le package est distribué auprès des clients.

Une fois l'application installée, Installshield propose une activité de mise à jour réalisée à partir de « *patches* » (partie d'une application), faisant référence au produit d'origine. En effet, seules les modifications apportées aux produits sont transmises au client pour mettre à jour son produit.

Les outils de type installateurs ont, généralement, un niveau d'abstraction assez limité. Le modèle de produit inclut au minimum la liste des fichiers à installer avec l'information sur la version et la plate-forme. Le modèle de site, souvent spécifique à la plate-forme ou limité, contient l'information sur la configuration du site, l'environnement utilisateur et le système de fichiers. De plus, le procédé de déploiement est assez rigide et peut difficilement être personnalisé. Certains de ces outils commerciaux supportent aussi l'activité de mise à jour.

4.3.2. Les gestionnaires de packages et l'outil RPM

Les gestionnaires de package sont des utilitaires de systèmes d'exploitation qui assistent les administrateurs dans la gestion des applications. Ils sont basés sur le concept de package et de dépôt (*repository*) de site qui stocke l'information sur l'état de chaque package installé. Un package contient un ensemble de fichiers et les méta-données décrivant le système. Ces méta-données sont fournies par le producteur, puis utilisées par le gestionnaire de packages et stockées sur le site client une fois le package installé.

En termes d'abstraction, le modèle de site est une collection de packages et un dépôt. Le modèle de produit fournit un ensemble riche d'attributs dans les méta-données du package, avec des paramètres qui peuvent modifier le comportement des procédures de déploiement.

Ce type d'outils offre cependant un support limité pour le déploiement à grande échelle et pour le déploiement des systèmes distribués.

RPM (*RedHat Package Manager*) [Rpm, ET96] est l'un de ces outils pour la gestion de packages sous Linux. Dans ce contexte, un package est une archive contenant tous les fichiers d'une application et des scripts de contrôle exécutés lors de l'installation ou de la désinstallation de l'application. Ce fichier contient aussi toutes les dépendances vers d'autres packages. Des commandes permettent d'installer, de mettre à jour ou de supprimer un package. Certaines permettent aussi d'effectuer des requêtes sur les packages (liste des packages installés, liste des fichiers contenus dans un package, etc.).

Il faut aussi souligner la différence entre l'installation et la mise à jour :

- La **mise à jour** écrase l'ancien package par le nouveau.
- L'**installation** conserve (si possible) l'ancienne version de l'application et installe la nouvelle en parallèle.

Enfin, des outils comme *apt-get* sur *RedHat* [RedH] permettent de faciliter la gestion des packages. Ces outils permettent notamment de prendre en charge les dépendances (résolution automatique) pour l'installation et la désinstallation ou de mettre à jour les packages installés avec les dernières versions existantes.

4.3.3. Les systèmes de gestion d'applications

Les systèmes de gestion d'application (comme Tivoli) étaient initialement utilisés pour gérer les réseaux locaux. Ainsi, ils sont capables de détecter les erreurs matérielles ou réseaux et de les transmettre à un central pour les étudier. Ces systèmes de gestion se sont ensuite intéressés à la gestion d'applications, et au procédé de déploiement.

Les systèmes de gestion d'applications sont destinés aux moyennes et grandes entreprises qui jouent à la fois le rôle de producteur et de consommateur de logiciels. Ils ont une architecture centralisée :

- Un producteur central valide les nouvelles versions disponibles.
- Une station de gestion contrôle les activités de déploiement et d'administration.

Les systèmes de gestion d'applications supportent toutes les activités du cycle de vie du déploiement. Ils permettent d'exécuter des sous-installations sur plusieurs machines et de coordonner l'installation d'applications distribuées.

Les méta-données sont stockées dans un dépôt. Elles concernent l'information sur les packages de produit, les configurations (statiques et dynamiques) du site et les spécifications du procédé de déploiement. De plus, ce type de système est capable de réaliser des inventaires, qui permettent d'introspecter un site client et de déterminer les applications installées. Enfin, les activités de déploiement peuvent être ordonnancées et optimisées en

fonction de la hiérarchie du site : ce type d'outils offre donc une vision entreprise du déploiement.

4.4. Discussion et synthèse sur les outils commerciaux pour le déploiement

Le monde industriel propose différents types d'outils, qui se focalisent chacun sur un ou plusieurs aspects du déploiement. Même si le paragraphe précédent a présenté brièvement quelques outils connus, nous avons choisi de présenter plus en détails des outils offrant une vision à grande échelle du déploiement.

Les outils pour le déploiement à grande échelle proposent de réaliser le déploiement (et l'administration) d'applications à distance, de manière centralisée. Il est alors nécessaire d'avoir une vision globale du grand nombre de machines cibles.

Tivoli est intéressant pour sa vision entreprise et permet la planification du déploiement. Il permet aussi de réaliser des inventaires des configurations. Il est alors possible de voir quelle(s) application(s) sont déjà installées, pour les mettre à jour ou en installer de nouvelles, par exemple. Le point faible est le manque de flexibilité au niveau du procédé de déploiement.

De son côté, Rembo Auto-Deploy utilise des sortes de stratégies de déploiement. En effet, il est possible d'associer des configurations spécifiques à des machines particulières. L'utilisation de stratégies mérite cependant d'être étendue pour offrir plus de possibilités à l'utilisateur. De plus, l'utilisation d'images de disques est une manière technique de définir ce qui est déployé sur une machine. D'une vision plus conceptuelle, un modèle de site peut décrire quelle(s) applications sont déployées sur chaque cible.

Un environnement de déploiement pour les entreprises doit donc fournir une vision globale de toutes les machines cibles de l'entreprise. Il doit aussi permettre de vérifier leur configuration et de déployer des applications à distance.

Pour offrir plus de possibilités à l'utilisateur, l'environnement doit aussi permettre de définir des stratégies de déploiement et de proposer un procédé de déploiement personnalisable. Celles-ci permettent de personnaliser le déploiement, tout en l'automatisant. En effet, le fait d'associer une configuration spécifique à un ensemble de machines permet non seulement de ne définir le déploiement qu'une seule fois (automatisation), mais aussi de définir une configuration pour chaque machine (personnalisation). Des stratégies de déploiement peuvent aussi permettrent d'ordonnancer un plan de déploiement. Ainsi, le procédé de déploiement n'est pas fixe et peut être différent selon les besoins de chaque entreprise.

5. SYNTHESE

Nous avons vu dans ce chapitre que plusieurs approches sont possibles pour aborder la problématique du déploiement. Nous avons tout d'abord présenté les approches par les modèles. L'ensemble de ces approches consiste à modéliser les informations liées au déploiement. Les modèles ainsi obtenus peuvent ensuite être spécialisés en fonction du type d'applications à déployer ou de l'environnement cible de déploiement.

Plusieurs approches sont proposées dans le milieu de la recherche. Chaque approche se focalise sur un ou quelques points particuliers du déploiement. Certaines se focalisent sur le

déploiement sensible au contexte, d'autres s'intéressent à la résolution de dépendances, etc. Ainsi, il est intéressant de collecter les points positifs et les axes de réflexion, afin de fournir un environnement de déploiement plus complet.

Enfin, les outils commerciaux (outils dédiés au déploiement d'applications *Windows*, par exemple) sont souvent plus spécifiques que les approches proposées par la recherche. Cependant, certains d'entre eux proposent des aspects intéressants, notamment ceux qui proposent une vision entreprise du déploiement, avec une vue plus globale de la cible de déploiement. Ces approches sont particulièrement intéressantes pour le déploiement à grande échelle.

Nous avons choisi de réaliser une approche par les modèles. En effet, cette approche, en plus d'être actuellement en pleine expansion, propose une solution générique. Elle permet d'abstraire l'ensemble des concepts nécessaires et de les spécialiser ensuite en fonction des applications à déployer ou des environnements cibles. Cette approche permet de réutiliser un maximum d'informations lorsque le type d'applications change. En effet, dans ce cas, seuls les outils spécifiques sont à réécrire.

De plus, nous utilisons des stratégies de déploiement, qui permettent de personnaliser le déploiement et de prendre en compte les contraintes imposées par l'entreprise. En effet, une configuration particulière peut être choisie pour une machine cible (ou un ensemble de machines cibles). Le procédé (plan) de déploiement peut aussi être différent selon les stratégies utilisées par chaque entreprise.

Notre approche est présentée dans la suite du document. Elle est introduite en deux étapes : d'abord, la modélisation des concepts liés au déploiement [Mer04] et leur mise en œuvre, puis, la définition et la mise en œuvre de stratégies de déploiement.

CHAPITRE 4 :
UN MODELE GENERIQUE POUR LE DEPLOIEMENT D'APPLICATIONS A GRANDE ECHELLE

Préambule :

Ce chapitre présente notre approche basée sur des modèles. Il s'agit de fournir une modélisation générique des données du déploiement. Ici, l'ensemble des concepts nécessaires au déploiement est utilisé pour définir un méta-modèle de déploiement. Chacun des concepts est détaillé, puis illustré dans un exemple à la fin du chapitre.

---

1. INTRODUCTION : NOTRE APPROCHE

Nous avons cité l'objectif de fournir un support automatisé pour le déploiement d'applications à grande échelle. L'automatisation du traitement du déploiement implique la modélisation de l'ensemble des concepts qui sont nécessaires.

De plus, nous souhaitons fournir un environnement générique qui soit indépendant :

- des applications que l'on déploie,
- de la configuration des sites cibles (machines finales) sur lesquelles sont déployées les applications,
- de l'environnement de l'entreprise dans lequel se passe le déploiement,
- des outils de déploiement utilisés (ces outils étant souvent spécifiques à un type d'application).

Pour fournir un tel environnement, il est indispensable de dégager l'ensemble des concepts spécifiques au déploiement, et communs à tous les déploiements, quelques soient les applications à déployer ou l'environnement cible de déploiement.

Pour cela, nous utilisons une approche dirigée par les modèles [BG01, Fav04, Omg03-a]. Ainsi, nous avons rassemblé l'ensemble des concepts liés au déploiement dans un méta-modèle de déploiement. Ce méta-modèle est formé de plusieurs parties contenant les différentes entités du déploiement que nous avons définies auparavant, à savoir :

- une partie liée à l'environnement de l'entreprise,
- une partie liée aux machines cibles,
- une partie liée aux applications à déployer.

Avec un tel méta-modèle, l'utilisateur peut ensuite créer son modèle spécifique en établissant les correspondances entre les concepts du méta-modèle de déploiement et ceux de son modèle spécifique. Ainsi, à partir du même méta-modèle, il est pratiquement possible de déployer tout type d'application (applications à base de composants ou applications type COTS [Car97]), dans n'importe quel environnement.

De plus, pour réaliser ces « correspondances » de concepts entre méta-modèle (données génériques) et modèles spécifiques, l'utilisateur peut réutiliser des modèles qui sont déjà définis et utilisés dans d'autres contextes. Ainsi, il peut, par exemple, réutiliser le modèle d'entreprise qui est utilisé pour la gestion administrative de l'entreprise, simplement en établissant les liens nécessaires.

Une telle approche dirigée par les modèles est particulièrement intéressante en ce qui concerne la réutilisabilité. En effet, une fois la partie générique définie, elle peut être utilisée dans tout type de contexte. Le travail à fournir alors est la définition (ou récupération) de la partie spécifique, en établissant les relations entre les données et outils spécifiques avec les concepts génériques.

La difficulté du travail, pour définir un méta-modèle de déploiement, est de s'abstraire de toute notion spécifique pour ne dégager que les concepts communs. Ainsi, par exemple, les concepts de service (dans des modèles comme OSGi [Osg03]) ou de composant (dans des modèles comme CCM [Ccm03] par exemple) sont à bannir. Pour les machines, il faut s'abstraire des notions spécifiques à un PC, à un ordinateur portable ou à une grappe. De même pour l'entreprise, dans le contexte du déploiement, il faut oublier les concepts de département, d'équipe ou de rôle pour ne conserver que la notion de groupe de machines. Nous verrons dans le chapitre suivant comment le méta-modèle est adaptable aux concepts spécifiques, en rattachant les concepts spécifiques aux concepts génériques.

Enfin, dans notre approche, nous avons choisi de créer un plan de déploiement rassemblant toutes les étapes à réaliser pour effectuer un déploiement de *n* applications sur *p* machines. Ce plan de déploiement doit aussi être indépendant de toute notion spécifique à la manière dont il va être utilisé (par exemple, des tâches *ANT*, un procédé, un langage, …).

Comme le montre la Figure 4.1, pour construire le plan de déploiement, différents concepts sont utilisés. Ensuite, chacun des « domaines » de concepts peut être personnalisé selon des modèles spécifiques.

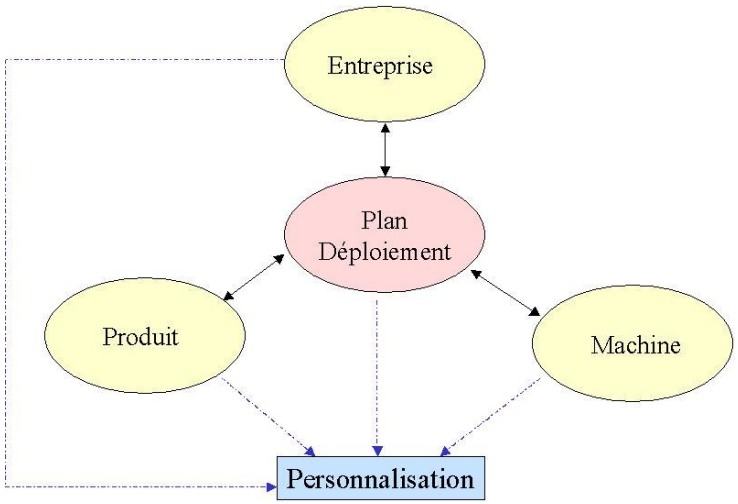

Figure 4.1 Personnalisation des concepts du méta-modèle de déploiement

Dans ce chapitre, nous définissons un méta-modèle de déploiement, avec tous les concepts à utiliser pour le déploiement.

2. MODELISATION DES INFORMATIONS LIEES AU DEPLOIEMENT

Pour structurer l'ensemble de l'information liée au déploiement, nous définissons un méta-modèle de déploiement (Figure 4.2).

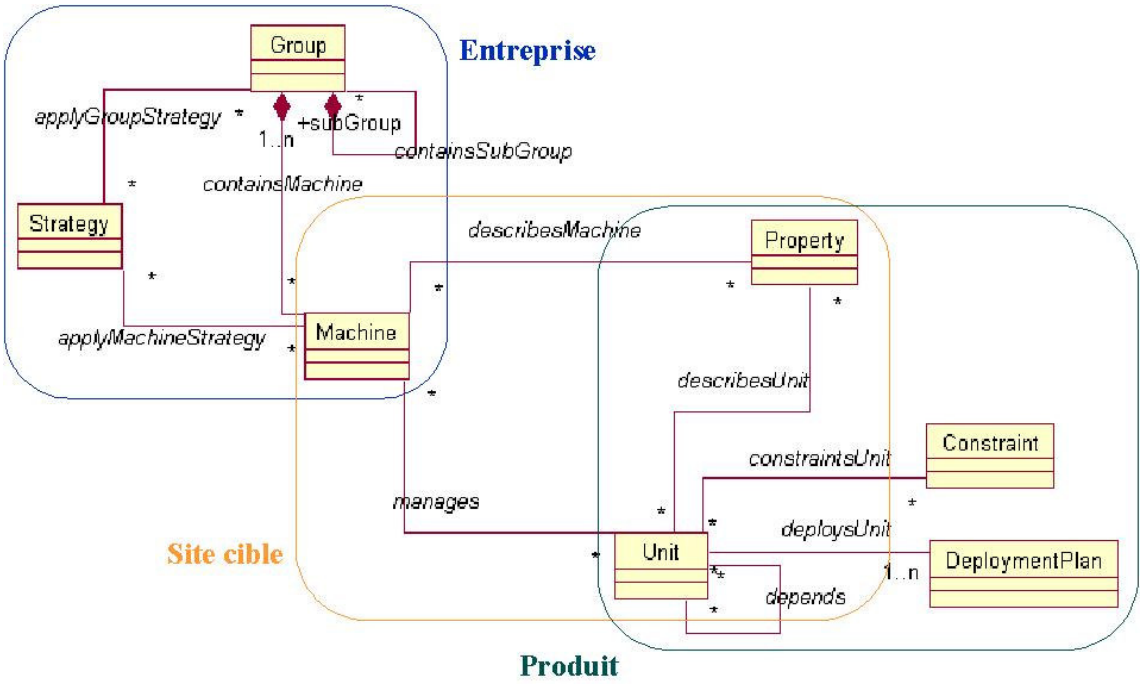

Figure 4.2 Méta-modèle de déploiement

Ce méta-modèle est présenté en trois parties distinctes, liées aux concepts d'entreprise (paragraphe 2.1), de site cible (paragraphe 2.2), puis de produit (paragraphe 2.3).

2.1. Concepts liés à la structure de l'entreprise

Un ensemble de concepts liés à la structure de l'entreprise est nécessaire au déploiement. Ces informations sont notamment utilisées lors de l'application des stratégies de déploiement et lors de la construction d'un plan de déploiement global. Ces informations sont modélisées par le méta-modèle de groupe.

Une entreprise est considérée comme un regroupement ou **groupe** d'entités. La structure est **hiérarchique**, c'est-à-dire qu'un groupe peut être composé de sous-groupes et/ou de machines. Ces groupes ont chacun des caractéristiques individuelles : machines appartenant au groupe et stratégies à appliquer. La Figure 4.3 illustre le méta-modèle des concepts liés à un groupe (et donc à une entreprise).

Un groupe est caractérisé par son nom et peut être composé de sous-groupes et/ou machines. Une machine peut appartenir à plusieurs groupes. De même, un groupe peut être sous-groupe de plusieurs groupes. Cependant il ne peut pas y avoir de relation cyclique entre les groupes : en représentant la structure hiérarchiques des groupes et sous-groupes de l'entreprise, on obtient un graphe non cyclique.

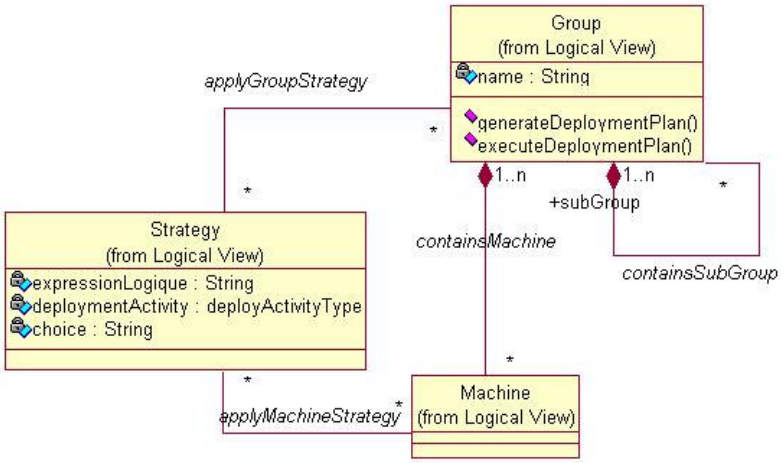

Figure 4.3 Méta-modèle de groupe

A ce niveau, il est possible de sélectionner une unité (ou un ensemble d'unités) de déploiement à déployer sur un sous-ensemble de machines (ou une seule machine). Il est aussi possible de générer le plan de déploiement qui permet de déployer une unité de déploiement sur l'ensemble des machines composant le groupe.

A un groupe sont appliquées des stratégies (*Strategy*) pour déployer des unités de déploiement sur les machines qui le composent. Ces stratégies permettent de définir des plans de déploiement. Différents types de stratégies peuvent être appliqués à différents moments du déploiement.

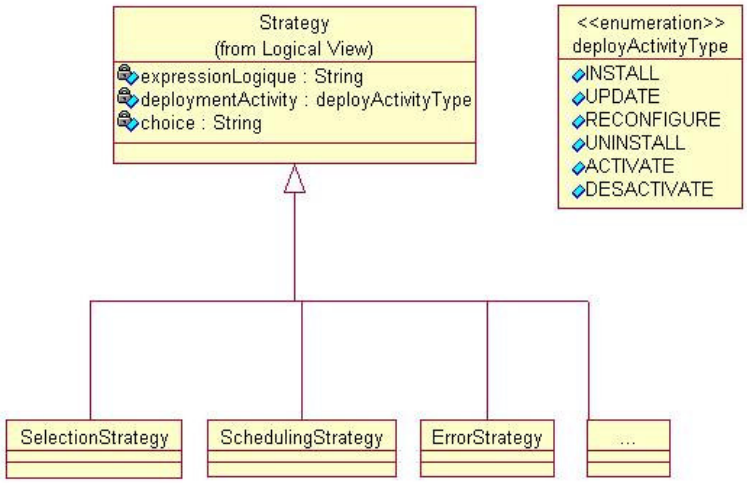

Figure 4.4 Raffinement de la classe *Strategy*

Une stratégie peut se spécialiser (Figure 4.4) en différents types de stratégies définis dans la suite du document. Les stratégies peuvent être associées à un groupe de machines ou à une machine seule. D'autres types de stratégies peuvent être ajoutés pour étendre le modèle, les types de stratégies définis dépendant des besoins d'une entreprise donnée.

Enfin, en considérant les concepts d'un modèle général d'entreprise (non spécialisé pour le déploiement), un groupe (*Group*) peut représenter une entreprise (Figure 4.5) ou une partie d'une entreprise, comme un département, une équipe de travail, un utilisateur (qui utilise plusieurs machines), un rôle d'utilisateurs ou d'autres sous-groupes de machines. Ceci implique une structure hiérarchique de l'entreprise. Pour le déploiement, peu importe la nature du groupe, il s'agit de définir l'ensemble de machines qui sera la cible d'un déploiement. C'est pourquoi **seul le concept de groupe** est utilisé dans le méta-modèle de déploiement.

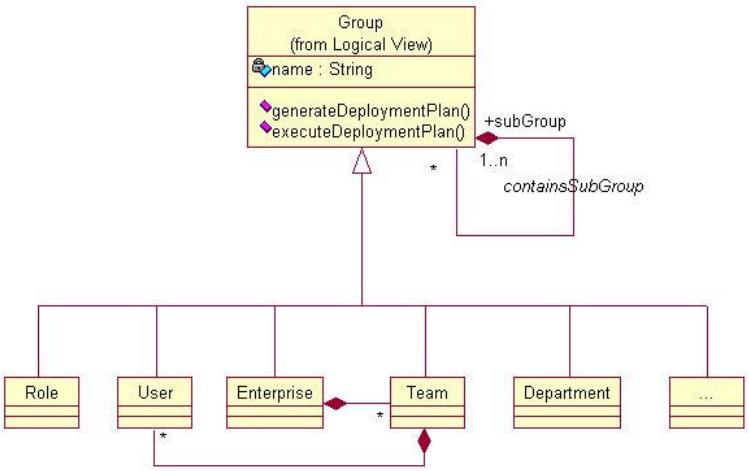

Figure 4.5 Vision « entreprise » de la classe *Group*

Chaque entreprise utilise ainsi sa propre structure. Elle définit alors l'ensemble des groupes qu'elle souhaite prendre en compte, ainsi que les relations et contraintes qui les lient. Par exemple (Figure 4.5), une entreprise (*Enterprise*) peut être composée d'un ensemble d'équipes (*Team*), elles-mêmes composées d'un ensemble d'utilisateurs (*User*).

2.2. Concepts liés à la description des machines

Un ensemble de concepts liés à la description des machines est nécessaire au déploiement. Ces informations décrivent les caractéristiques logicielles et matérielles des machines :

- Les **caractéristiques matérielles** sont décrites à l'aide d'un ensemble de propriétés. Elles indiquent les caractéristiques matérielles de la machine, comme le processeur, la mémoire, l'espace disque, le type de carte vidéo, …
- Les **caractéristiques logicielles** décrivent l'état de la machine, c'est-à-dire l'ensemble des unités déployées sur la machine. D'autres caractéristiques logicielles peuvent aussi être décrites avec des propriétés (le système d'exploitation, par exemple).

Ces informations sont utilisées pour choisir des versions d'application (unités de déploiement) adaptées (le plus possible) à la configuration des machines et ne déployer que des unités qui fonctionneront correctement sur les sites clients.

La Figure 4.6 illustre le méta-modèle des concepts liés à la description des machines.

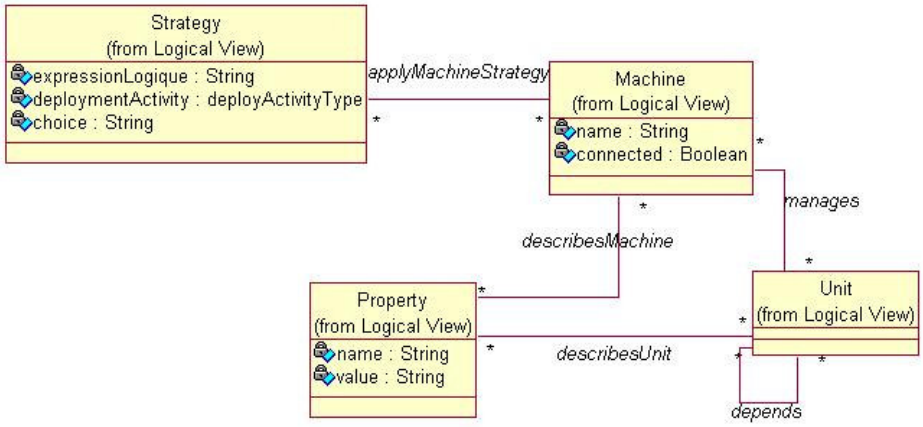

Figure 4.6 Méta-modèle de machine

Un ensemble de propriétés (*Property*) permet de décrire la machine. Par exemple, des propriétés décrivent le nombre de sorties vidéo de la machine, son espace disque disponible ou son système d'exploitation. Les descriptions permettent de savoir si les contraintes imposées par une unité de déploiement pourront être satisfaites (et donc si l'unité pourra être déployée sur le site client).

Un ensemble de stratégies (*Strategy*) permet d'indiquer quelles unités de déploiement pourront être déployées sur le site client et sous quelles conditions elles seront déployées. Les stratégies peuvent se spécialiser dans les différents types définis précédemment (Figure 4.4).

Une machine gère un ensemble d'unités de déploiement (*Unit*) : ce sont les unités qui sont déjà déployées.

En réalité, deux types de machines peuvent se distinguer : les serveurs d'applications et les sites clients (Figure 4.7).

Les machines qui sont des serveurs d'applications fournissent des unités à déployer sous forme packagée. Un package contient toutes les données et l'information nécessaires au déploiement d'une unité. Les machines qui sont des sites clients sont des machines où des unités sont déployées.

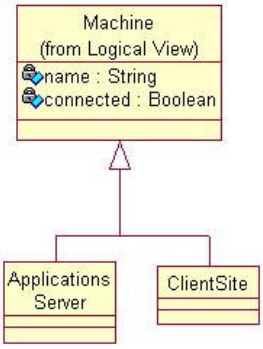

Figure 4.7 Raffinement de la concept *Machine*

Dans la suite nous ne considérons que les machines clientes, c'est à dire celles sur lesquelles les unités sont déployées. En effet, les serveurs d'applications servent simplement au stockage d'unités à déployer et ne concernent pas directement la problématique du déploiement (par exemple, les dépendances ne sont pas résolues) : ils sont avant tout utilisés comme dépôt d'unités.

2.3. Concepts liés à la description du produit à déployer

Il faut tout d'abord préciser qu'une application passe par différentes phases de vie. Cette partie décrit tout d'abord les différences entre ces phases, puis présente la modélisation nécessaire au déploiement.

2.3.1. Différentes phases dans la vie d'une application

Une application (ou produit) passe par différentes phases de vie :

- sa construction,
- son déploiement,
- son exécution.

Chacune de ces phases de vie implique une vision différente du produit et nécessite une modélisation spécifique. Ainsi, lorsque l'application est construite (en fonction des besoin de l'entreprise), une vision en termes de modules (ou composants) est nécessaire. Cette vision permet de décider quels composants fournir à l'entreprise en fonction des services qu'elle souhaite.

A l'issue de cette phase, l'application est *packagée* avec l'information nécessaire. On obtient alors une unité de déploiement, qui est le grain du déploiement. C'est cette vision que nous utilisons et qui est défini ci-dessous. Une unité de déploiement peut représenter une application complète, mais aussi plusieurs applications, ou même seulement une partie de l'application. Ce qui importe ici, c'est de savoir comment la déployer.

Une fois déployée, l'unité de déploiement est « transformée » en composants ou services qui vont s'exécuter. En prenant l'exemple d'OSGi, les unités de déploiement sont des *bundles*, mais ce sont des *services* qui vont s'exécuter. Il s'agit ici de repérer l'architecture de l'application à l'exécution (architecture dynamique).

Ces différentes phases de vie d'une application proposent des visions différentes et exigent des modélisations distinctes. Cependant, il est nécessaire de connaître les correspondances entre les données utilisées dans ces différentes phases. Par exemple, pour reprendre l'exemple d'OSGi, c'est un *service* qui peut être demandé à tout moment, mais ce sont des *bundles* qui sont déployés : il faut donc être capable de dire quel(s) *bundle(s)* fourni(ssen)t ce *service*.

Cette correspondance peut, par exemple, se faire via des descripteurs qui permettent de définir le produit dans ces différentes phases de vie. Ils permettent alors de créer les liens entre les différents modèles. Dans notre travail, nous nous limitons à la phase de déploiement d'un produit, c'est pourquoi nous traitons des unités de déploiement.

2.3.2. Modélisation d'une unité de déploiement

Un ensemble de concepts liés aux unités de déploiement est nécessaire au déploiement. Ces informations permettent notamment de décrire les unités et d'exprimer les contraintes de déploiement qu'elles imposent.

La Figure 4.8 illustre le méta-modèle des concepts liés à la description des produits. Une unité de déploiement (*Unit*) représente un produit (version d'une application).

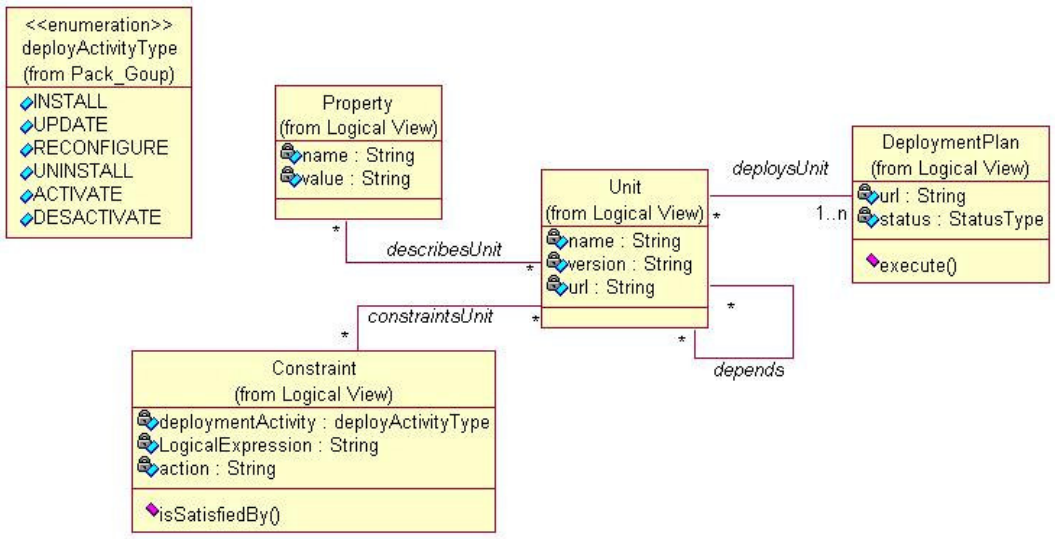

Figure 4.8 Méta-modèle d'unité de déploiement

Une unité est caractérisée par son nom (nom de l'application), son numéro de version et ses dépendances. Ici, nous ne prenons pas en compte la configuration des unités. Cette activité est réalisée en amont du déploiement, lorsque l'entreprise énonce ses besoins au producteur. Lorsque plusieurs versions d'une application (unités ayant le même nom) sont disponibles, nous considérons qu'elles répondent toutes aux besoins énoncés par l'entreprise et que c'est l'une d'entre elles qui doit être déployée sur chaque machine en fonction des contraintes et des stratégies.

Dans les modèles à composants, on utilise une relation de composition : un composant composite peut être composé de composants de base. Ici, la notion de composition n'apparaît pas au niveau des unités. En effet, deux cas peuvent se produire :

- La composition est interne à l'unité de déploiement et il n'est pas nécessaire pour le déploiement de connaître cette composition.
- Il existe une dépendance explicite avec une autre unité. Dans ce cas, il s'agit d'une relation de dépendance, et non de composition.

Chaque unité est aussi caractérisée par un ensemble de propriétés (*Property*) et de contraintes (*Constraint*) qui lui sont spécifiques.

Un ensemble de propriétés permet de décrire une unité et de la sélectionner pour un déploiement donné. Par exemple, une propriété peut indiquer que l'implémentation est réalisée en java (<*Implémentation, Java*>).

Un ensemble de contraintes (*Constraint*) permet d'indiquer quelles propriétés doit avoir un site client pour accueillir l'unité de déploiement. Pour savoir si un site cible respecte les contraintes imposées par l'unité de déploiement, on étudie les propriétés du site en relation avec les contraintes de l'unité.

De plus, un plan de déploiement (*DeploymentPlan*) indique comment déployer l'unité. Les plans de déploiement peuvent se spécialiser selon les différentes activités du cycle de vie du déploiement (Figure 4.9).

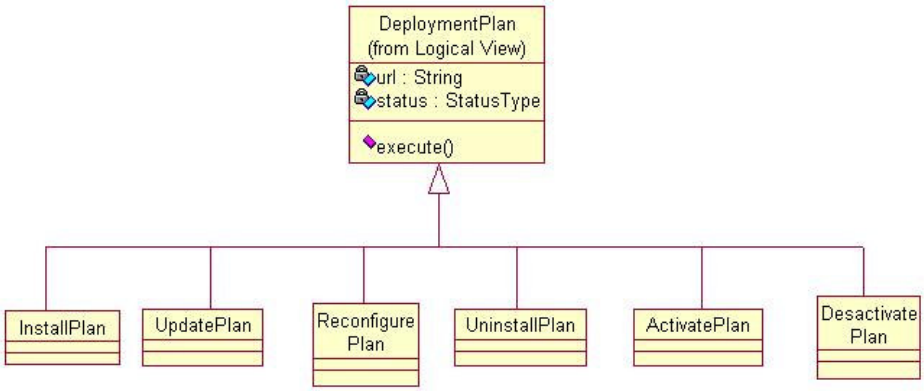

Figure 4.9 Raffinement de la classe *DeploymentPlan*

Les plans de déploiement liés à une unité de déploiement sont ensuite composés pour créer un plan de déploiement global permettant de déployer une unité sur un groupe de machines d'une entreprise.

De plus, une unité peut dépendre d'autres unités. Une unité dépend d'une autre unité lorsqu'elle en a besoin pour fonctionner correctement. Les différents types de dépendances sont exposés dans le paragraphe suivant.

3. DEPENDANCES ENTRE UNITES DE DEPLOIEMENT

Nous avons vu dans la partie précédente qu'une unité de déploiement peut dépendre d'une autre unité. Cela signifie qu'elle a besoin de cette deuxième unité pour fonctionner.

Par hypothèse, nous considérons que les dépendances d'une unité sont identifiées précisément : une version d'une application dépend d'une version donnée d'une autre application. Pour reprendre les concepts modélisés ci-dessus, une unité de déploiement donnée dépend d'une autre unité de déploiement donnée.

Pour chaque dépendance, des stratégies (ou politique) de déploiement différentes peuvent être appliquées. Celle-ci désignent si les dépendances doivent aussi être déployées ou non. Les stratégies indiquent aussi à quel moment les unités dépendantes peuvent/doivent être déployées.

Plusieurs types de dépendances peuvent être identifiés [RDU04] et traitées de manière différentes. Parmi les types de dépendances, nous considérons :

- les dépendances obligatoires,
- les dépendances de conflit,
- les dépendances sélectives,
- les dépendances de remplacement,
- les dépendances facultatives.

Les paragraphes suivants détaillent ces différents types de dépendances.

3.1. Les dépendances obligatoires

On dit qu'une unité de déploiement *A* dépend (dépendance obligatoire) d'une unité *B*, lorsque *A* a besoin de *B* pour s'exécuter correctement.

Dans ce cas, plusieurs stratégies de déploiement peuvent être appliquées. Par exemple, l'entreprise peut décider de déployer les dépendances en même temps que l'unité de déploiement, c'est-à-dire déployer *B* en même temps que *A*. L'entreprise peut aussi choisir de ne déployer que *A* et d'attendre qu'à l'exécution *A* utilise *B* pour déployer *B*, en espérant que *B* ne soit pas nécessaire tout de suite ou alors qu'il s'agisse d'une dépendance dont *A* n'a pas systématiquement besoin pour s'exécuter.

3.2. Les dépendances de conflit

Il existe un conflit entre deux unités de déploiement *A* et *B*, quand les deux unités ne peuvent pas cohabiter sur la même machine. Dans ce cas, la relation de dépendance est symétrique : une unité *A* est en conflit avec une unité *B* et *B* est en conflit avec *A*.

Dans ce cas, l'entreprise peut décider de ne déployer que l'une des deux unités sur chacune de ses machines. Ainsi, si *A* est déjà déployée, alors le déploiement de *B* ne pourra pas être réalisé (et inversement, si c'est *B* qui est déjà déployée). Le choix de déployer, malgré tout, les deux unités sur une même machine peut aussi être fait, s'il s'agit d'un conflit d'exécution et que les deux unités ne seront pas utilisées en même temps.

3.3. Les dépendances sélectives

Dans le cas des dépendances sélectives, une ou plusieurs unités de déploiement peuvent être choisies. Ainsi, une liste d'unités est disponible et le déploiement de l'un des éléments de cette liste résout la dépendance. Cela signifie que la dépendance peut être résolue de plusieurs manières.

Selon les préférences de l'entreprise ou de l'utilisateur, le choix de l'unité pourra, par exemple, se porter vers une unité implémentée en java, vers l'unité qui occupe le moins d'espace disque, etc.

3.4. Les dépendances de remplacement

Une unité de déploiement *A* a une dépendance de remplacement vis à vis d'une unité *B*, lorsque *A* peut être remplacée par *B*. Dans ce cas, cela signifie que, **pour cette utilisation**, les unités ont les mêmes fonctionnalités ou fournissent les mêmes services. Il faudra alors déployer l'une ou l'autre des unités (*A* ou *B*).

Il faut noter que ce type de dépendance peut être remplacé par une dépendance sélective au niveau supérieur.

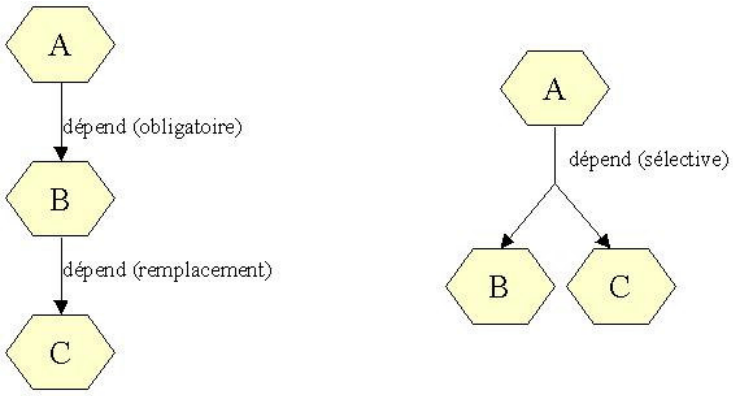

Figure 4.10 Dépendance de remplacement et dépendance sélective

La Figure 4.10 montre ainsi deux manières de modéliser les mêmes dépendances effectives entre unités de déploiement. Dans la partie de gauche, *A* dépend de *B*, qui dépend (dépendance de remplacement) lui-même de C. Si le déploiement de l'unité *A* (et de ses dépendances) est demandé, les unités *A* et *B* ou alors *A* et *C* seront donc déployées. Pour le schéma de la partie droite, *A* a une dépendance sélective vers la liste d'unités composée de *B* et *C*. Cela signifie que l'une des unités *B* ou *C* sera déployée pour que *A* fonctionne correctement. On se retrouve alors avec les mêmes cas de déploiement que dans l'autre schéma.

3.5. Les dépendances facultatives

Une unité de déploiement *A* à une dépendance facultative avec l'unité *B*, lorsque *A* peut fonctionner avec ou sans *B*. Dans ce cas, le déploiement de l'unité de dépendance est facultatif. Par exemple, une dépendance vers une unité de documentation ou vers un module optionnel du logiciel peut être qualifiée de facultative : le produit peut fonctionner sans ces unités, mais l'utilisateur aura plus de possibilités si elles sont présentes.

Dans ce cas, l'entreprise peut décider de toujours déployer ce type de dépendances pour offrir le maximum d'options aux utilisateurs. Elle peut aussi choisir de ne jamais les déployer en déployant des versions minimales sur chaque machine. Elle peut aussi appliquer des stratégies plus complexes, comme de déployer ces dépendances dans le cas où l'espace disque est suffisant, par exemple.

4. CONTRAINTES ET STRATEGIES DE DEPLOIEMENT

Dans la modélisation, nous avons choisi d'associer des **contraintes** (*Constraint*) aux **unités de déploiement** et des **stratégies** (*Strategy*) aux **entités de l'entreprise**. De plus, des **propriétés** (*Property*) permettent de décrire les **unités**, mais aussi les **machines cibles**. Celles-ci sont donc associées aussi bien au niveau des produits (unités) qu'au niveau des sites (machines) de l'entreprise.

Les contraintes d'une unité représentent alors l'ensemble des caractéristiques que doit avoir une machine cible pour pouvoir l'accueillir. Il s'agit ici de vérifier si l'unité peut être déployée (suffisamment d'espace disque par exemple) et si elle pourra s'exécuter et fonctionner correctement (capacité mémoire suffisante par exemple).

Ces contraintes sont connues par le producteur de l'unité de déploiement et doivent donc être définies par lui. C'est pourquoi nous avons choisi d'associer les contraintes aux unités de déploiement.

Ces contraintes sont ensuite mises en correspondance avec les propriétés des machines cibles de l'entreprise. Cette mise en correspondance permet de définir si les propriétés de la machine vérifient ou non les contraintes imposées par une unité de déploiement, et donc si celle-ci peut-être déployée ou non.

De l'autre côté, nous avons associé les stratégies aux entités de l'entreprise. Une stratégie permet de définir un choix de déploiement : soit une préférence d'un type d'unités (unités implémentées en java par exemple), soit un ordre de déploiement, etc. Ces choix sont propres aux entreprises et/ou aux utilisateurs de l'entreprise. Le producteur ne peut en aucun cas décider des politiques de l'entreprise. C'est pourquoi nous avons fait le choix d'associer le concept de stratégie aux concepts de groupe et de machine.

L'entreprise peut aussi souhaiter appliquer une stratégie seulement pour un type précis d'unité de déploiement. Par exemple, une stratégie peut exprimer qu'il faut déployer les unités implémentées en java, d'abord sur la machine M1, puis sur la machine M2. Cette stratégie ne doit être appliquée que sur un ensemble précis d'unités de déploiement : celles implémentées en java. Pour exprimer de telles restrictions, une sorte de garde (un filtre) est associée à chaque stratégie et permet de définir quand l'appliquer.

Les stratégies sont présentées en détails dans le Chapitre 6. Les contraintes sont, quant à elles, présentées dans la partie suivante.

5. CONTRAINTES D'UNE UNITE DE DEPLOIEMENT

Une unité de déploiement ne peut pas toujours fonctionner correctement sur n'importe quel type de machine. Par exemple, un produit prévu pour fonctionner uniquement sous *Linux* ne peut pas fonctionner correctement sur une machine sous *Windows*. Il faut donc exprimer un certain nombre de contraintes imposées par l'unité de déploiement.

5.1. Définition d'une contrainte

Pour déterminer si une contrainte est vérifiée ou non par la configuration d'une machine, on utilise les propriétés décrivant la machine. Par exemple, si une unité de déploiement ne

fonctionne que sous *Windows XP*, on vérifie la propriété de la machine indiquant le système d'exploitation. En fonction de sa valeur, la contrainte est, ou non, vérifiée et l'unité de déploiement peut, ou non, être déployée. La définition d'une contrainte va donc utiliser des valeurs de propriétés.

De manière générale, nous avons choisi de représenter une contrainte par un triplet *<ActivitéDeDéploiement, ExpressionLogique, Action>*. L'expression logique (*ExpressionLogique*) est une expression utilisant des noms et valeurs de propriétés que doit avoir une machine cible éventuelle. Si cette expression est vérifiée (c'est-à-dire son évaluation donne *vrai*), la contrainte est vérifiée et l'action (*Action*) devra être réalisée **après le déploiement** de l'unité associée.

De plus, une même unité de déploiement peut être utilisée au cours de plusieurs activités du cycle de vie du déploiement. Par exemple, la même unité peut être utilisée pour une première installation du produit, ou pour une mise à jour du produit déjà déployé. Dans ce cas, certaines contraintes peuvent n'être associées qu'à l'une de ces activités. Dans le triplet précédent, le champ *ActivitéDeDéploiement* indique ainsi lors de quelle activité du déploiement la contrainte doit être prise en compte. Par exemple, si une contrainte indique *INSTALL* pour le champ *ActivitéDeDéploiement*, elle ne sera prise en compte que lors d'une installation de l'unité.

5.2. Différents types de contraintes

Les contraintes associées aux unités de déploiement peuvent exprimer des choses complètement différentes. Dans le modèle de l'OMG [Omg04], les contraintes ont été classées en plusieurs catégories :

- Les **contraintes de type *Quantité*** expriment une quantité qui doit être décrémentée (de un) lors du déploiement. Par exemple, considérons une unité de déploiement qui nécessite une sortie vidéo. Avant le déploiement, le nombre de sorties vidéo disponibles sur la machine cible doit donc être supérieur ou égal à 1. Après le déploiement, le nombre de sorties vidéos disponibles sur la machine cible est décrémenté de un.
- Les **contraintes de type *Capacité*** expriment une capacité qui diminue lors du déploiement. La machine cible doit donc pouvoir fournir cette capacité au moment du déploiement pour pouvoir recevoir l'unité. Par exemple, une unité nécessite un espace disque disponible de 14 Mo. Avant le déploiement, cette capacité disponible sur la machine cible doit donc être supérieure ou égale à cette même valeur. Après le déploiement, l'espace disque disponible sur la machine cible est décrémenté de cette valeur.
- Les **contraintes de type *Minimum*** expriment un minimum nécessaire lors du déploiement. Il s'agit ici d'indiquer quelle valeur minimum doit avoir l'une des propriétés de la machine cible pour recevoir l'unité de déploiement. Par exemple, une unité nécessite une mémoire d'au moins 256 Mo. Lors du déploiement, on vérifie que la mémoire de la machine cible est bien supérieure ou égale à la valeur indiquée. Dans ce cas, aucune action ne doit être exécutée, il s'agit simplement de vérifier une pré-condition au déploiement.
- Les **contraintes de type *Maximum*** expriment un maximum nécessaire lors du déploiement. Le fonctionnement des contraintes exprimant un maximum est

complètement symétrique de celui des contraintes exprimant un minimum. Par exemple, on utilise une telle contrainte pour tester la vitesse CPU d'une machine cible qui doit être inférieure ou égale à 700 MHz.

- Les **contraintes de type *Attribut*** expriment une caractéristique que doit avoir une machine cible pour recevoir l'unité de déploiement. Par exemple, une unité de déploiement peut exprimer qu'elle ne fonctionne qu'avec *WindowsXP*. Dans ce cas, une machine cible potentielle devra avoir l'une de ces propriétés indiquant que son système d'exploitation est bien *WindowsXP*.
- Les **contraintes de type *Sélection*** expriment une caractéristique que doit avoir une machine cible pour recevoir l'unité de déploiement, et pour laquelle plusieurs valeurs sont possibles. Par exemple, une unité de déploiement peut exprimer qu'elle fonctionne sous *WindowsXP* ou *Windows2000*.

Si l'on met en parallèle ces différents types de contraintes avec le triplet que nous avons défini précédemment, nous pouvons remarquer que dans le cas des contraintes de l'OMG, certaines vont imposer une action (quantité et capacité), alors que d'autres n'expriment qu'une sorte de pré-condition au déploiement (minimum, maximum, attribut, sélection). Cela implique que chaque type de contrainte devra être traité différemment.

De plus, certaines distinctions sont intéressantes au niveau « sémantique », mais a priori inutile au niveau du déploiement. Par exemple, la différence entre un minimum et un maximum est un simple changement du sens de l'inégalité à exprimer. De la même manière, les contraintes de quantité et capacité sont relativement proches, puisqu'elle utilisent toutes les deux des valeurs numériques, en exprimant des inégalités entre elles et en définissant des opérations à réaliser.

Pour le déploiement, il s'agit simplement de connaître quelle(s) valeur(s) vérifier et quelle(s) action(s) exécuter. Nous pouvons alors distinguer deux catégories de contraintes :

- **Les contraintes utilisées pour définir une pré-condition et une action à réaliser :** Des contraintes faisant intervenir une valeur numérique sont utilisées pour comparer cette valeur à celle d'une propriété d'une machine cible. Elles peuvent aussi servir à décrémenter la valeur de cette même propriété.
- **Les contraintes utilisées pour définir une pré-condition :** Des contraintes faisant intervenir des ensembles de valeurs sont utilisées pour vérifier si la valeur d'une propriété d'une machine cible appartient ou non à cet ensemble. Des contraintes faisant intervenir une valeur numérique peuvent aussi être utilisées pour exprimer une comparaison.

De manière plus générale, l'utilisateur peut décider de vérifier des pré-conditions plus complexes, et / ou des actions non triviales à réaliser. C'est pourquoi, il est important de ne pas restreindre les contraintes associées aux unités de déploiement par quelques types prédéfinis. C'est pour cela que nous avons choisi de rester générique pour la définition d'une contrainte. Ainsi, l'expression logique utilisée dans notre triplet permet d'exprimer tout type de pré-condition. L'action, quant à elle, permet d'effectuer n'importe quelle suite d'instructions. Enfin, le fait d'associer une activité de déploiement aux contraintes permet d'exprimer plus de cas.

Nous verrons dans le chapitre suivant comment ceci est réalisable grâce à la définition d'un langage de contraintes.

5.3. Cumul de contraintes et simulation de déploiement

Chacune des contraintes d'une unité de déploiement doit tout d'abord être vérifiée individuellement. Mais, si toutes les contraintes sont vérifiées individuellement, il faut ensuite vérifier que le « cumul » de toutes ces contraintes permet toujours le déploiement de l'unité. Ceci est particulièrement vrai dans le cas du déploiement d'une unité avec des dépendances.

En effet, l'unité et chacune de ses dépendances peuvent accéder aux mêmes propriétés de la machine cible. Ainsi, par exemple, la propriété représentant l'espace disque disponible pourra être consultée plusieurs fois, de manière individuelle. Mais en cumulant tous les espaces disque nécessaires (indiqués par chacune des unités), l'espace disque disponible de la machine peut ne plus être suffisant. C'est pourquoi, il est nécessaire d'effectuer une simulation des opérations réalisées par rapport aux contraintes. Il s'agit d'une simulation du déploiement pour vérifier que toutes les opérations à effectuer peuvent être réalisées, non seulement indépendamment, mais aussi toutes réunies. Cette simulation permet de s'assurer de la faisabilité du déploiement.

Il faut ajouter que nous ne traitons ici que les informations « statiques », liées au déploiement. En effet, comme nous nous sommes particulièrement intéressés à la première installation des unités sur les machines, il n'est pas encore nécessaire de traiter les aspects « dynamiques » (comme la capacité mémoire disponible à un instant donné, par exemple). Les aspects dynamiques sont nécessaires pour les activités du déploiement comme l'activation.

Le traitement des aspects dynamiques implique l'utilisation d'outils spécifiques permettant d'introspecter les caractéristiques d'un site. Dans le cas des aspects statiques, seul le modèle de site doit être interrogé.

6. PLAN DE DEPLOIEMENT

Dans notre approche, nous avons choisi d'utiliser un plan de déploiement pour définir quelles étapes réaliser pour effectuer une activité de déploiement.

6.1. Définition d'un plan de déploiement

Deux types de plans de déploiement peuvent être distingués : le plan de déploiement **unitaire**, et, le plan de déploiement **global**.

6.1.1. Plan de déploiement unitaire

Un plan de déploiement unitaire est associé à chaque unité de déploiement. Il permet d'établir la liste des instructions à réaliser pour déployer l'unité sur une machine. Ce plan est spécifique à une unité : il est fourni par le producteur puisque c'est le seul à avoir la connaissance sur les étapes à réaliser pour déployer son produit. Chaque unité peut être associée à plusieurs plans de déploiement unitaires, mais un seul plan par activité de déploiement peut être associé. Ainsi, une unité a au maximum un plan pour son installation, un plan pour sa mise à jour, un plan pour sa désinstallation, etc.

Un tel plan contient un lien vers les instructions à réaliser pour déployer une unité sur une machine. Les instructions peuvent, par exemple, être des instructions de copie de fichiers ou d'exécution de commandes. Le producteur peut laisser ouvert certains choix de déploiement comme le nom des répertoires à utiliser (chemins de déploiement). Ces paramètres seront instanciés lors du déploiement de l'unité sur chaque machine cible en fonction de la liste de ses propriétés (*Property*). Selon sa spécialisation, les instructions de déploiement peuvent prendre plusieurs formes comme, par exemple, un langage à interpréter ou un procédé à exécuter.

6.1.2. Plan de déploiement global

Lorsque l'entreprise a accès à des unités de déploiement, l'administrateur peut demander leur déploiement sur l'un des groupes de l'entreprise. Pour cela, il faut générer automatiquement le plan de déploiement global. Un plan de déploiement global concerne donc le déploiement de *n* unités sur *p* machines. Les différents plans de déploiement unitaires, concernant un couple <unité de déploiement, machine cible>, servent alors de briques de base pour construire le plan de déploiement global.

Il faut noter que le choix de réaliser un plan de déploiement peut entraîner de nouveaux cas d'erreur. En effet, l'environnement cible de déploiement peut être modifié entre le moment où le plan a été construit, et le moment où le plan est exécuté. Il peut s'agir du déploiement de nouvelles unités, d'une modification de configuration d'une machine, etc. Il faut alors penser à proposer un mécanisme de gestion des erreurs pour faciliter le travail de l'utilisateur dans de tels cas. Pour cela, nous verrons plus tard que des stratégies peuvent être utilisées.

En revanche, l'utilisation d'un plan de déploiement permet de repérer avant le début d'un déploiement si le déploiement pourra être accompli correctement (si l'environnement ne subit pas de modification). Ainsi une impossibilité de réaliser entièrement un déploiement est détectée avant de commencer concrètement le déploiement.

Par exemple, si aucune des unités de déploiement n'est compatible avec la configuration de l'une des machines cibles, le déploiement ne commencera pas. Dans le cas où aucun plan global n'est établi, un déploiement partiel peut être réalisé sur plusieurs machines avant de s'apercevoir qu'il y a un problème au niveau de l'une des cibles. Dans ce cas, il faudrait défaire plusieurs opérations réalisées.

L'utilisation d'un plan de déploiement permet donc de n'exécuter que les déploiements « théoriquement » corrects. Cette technique permet aussi de réaliser une sorte de simulation du déploiement.

6.2. Modélisation d'un plan de déploiement global

Un plan de déploiement global contient l'ordonnancement des différents plans de déploiement unitaires, fournis avec les unités à déployer. Le plan de déploiement global reflète le résultat de l'application des contraintes (imposées par les unités) et des stratégies de déploiement (imposées par l'entreprise). Avec le plan global, il s'agit d'ordonner les différents plans unitaires en respectant les contraintes et stratégies de déploiement. L'ordre ainsi obtenu peut être partiel ou total. En effet, l'ordre peut ne pas être entièrement déterminé. Le plan de déploiement peut donc présenter différents degrés de libertés, et donc contenir des choix. Ces degrés peuvent concerner l'ordre de déploiement mais aussi la sélection des unités à déployer (version(s) à déployer sur une machine, dépendances sélectives, …). L'ordre sera totalement

déterminé lors de l'exécution du plan de déploiement, par exemple en appliquant des stratégies complémentaires de déploiement. En effet, des stratégies peuvent être définies pour la phase d'exécution du plan. Ces stratégies pourront par exemple prendre en compte la disponibilité des machines ou les connexions réseaux.

Les degrés de liberté accordés au plan de déploiement dépendent des choix de mise en œuvre et d'implémentation choisis par l'entreprise. En effet, certaines entreprises peuvent préférer faire les choix nécessaires le plus tard possible. D'autres, au contraire, préfèrent tout fixer avant le début de l'exécution du plan de déploiement. Dans ce deuxième cas, le plan de déploiement global obtenu est alors reproductible : il peut être exécuté plusieurs fois et donnera toujours la même liste d'étapes (dans le même ordre). Dans le premier cas, plusieurs exécutions du plan de déploiement obtenu peuvent donner des résultats différents. En effet, si l'ordre de déploiement est partiel, la liste des déploiements unitaires pourra s'effectuer dans un ordre différent. De la même manière, si des choix d'unités restent à faire, deux exécutions du plan de déploiement pourront donner le déploiement d'unités différentes.

Nous pouvons maintenant présenter une modélisation d'un plan de déploiement global en prenant en compte les objectifs que nous venons de citer, à savoir :

- refléter le résultat de l'application des contraintes et stratégies de déploiement,
- ordonner les déploiements unitaires (couples <unité de déploiement, machine>),
- indiquer les choix possibles d'unités à déployer (versions d'application et/ou dépendances sélectives).

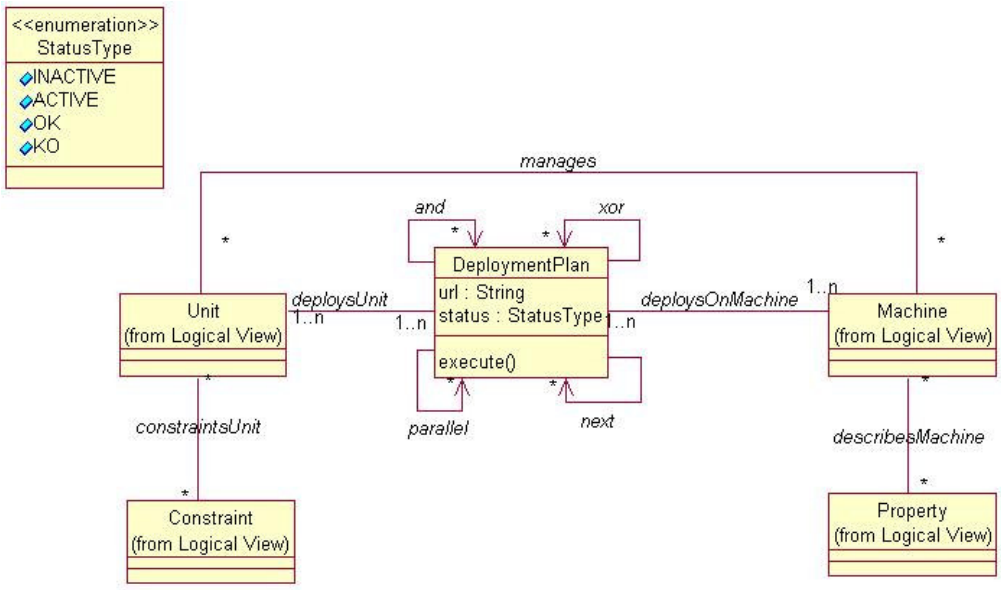

Figure 4.11 Modèle de plan de déploiement global

Le plan de déploiement unitaire, que nous avons défini précédemment, est la brique de base du plan de déploiement global : il est donc au centre de la modélisation (Figure 4.11). Le plan unitaire est caractérisé par l'URL qui permet d'accéder aux instructions de déploiement

(quelque soit leur forme). La méthode *execute()* accède donc à ces instructions pour lancer le déploiement de l'unité.

Chacun de ces plans permet de déployer une unité sur une machine. Chaque unité impose ses propres contraintes, et chaque machine a sa propre configuration (définie par ses propriétés, et les unités qui sont déjà déployées). Le fait de conserver ces liens permet d'identifier l'état de l'environnement de déploiement au moment où le plan global est construit. Ainsi, avant l'exécution du plan, une comparaison entre l'état indiqué par le plan et l'état courant peut être effectuée, pour vérifier si le plan peut s'exécuter correctement, ou risque d'engendrer des erreurs dues à l'évolution de l'environnement de déploiement.

Une fois ces associations réalisées, il reste à exprimer les relations d'ordre (partiel ou total) et les choix qui peuvent encore être faits entre ces différents déploiements unitaires. Il faut pouvoir exprimer quel plan de déploiement doit être le prochain à être exécuté (relation *next*), ou quel(s) plan(s) doi(ven)t être exécuté(s) en même temps (relation *parallel*). Pour les relations d'ordre partiel et les choix entre unités, les relations *and* et *xor* permettent de dire :

- *xor* : un des plans reliés par cette relation doit être exécuté,
- *and* : tous les plans reliés par cette relation doivent être exécutés selon un ordre indéfini.

Une relation pourrait aussi être définie pour augmenter les possibilités fournies par ces deux dernières relations. En effet, une relation permettant de choisir *de* déployer sur *n* machines parmi un ensemble de *p* ($p > n$) est intéressante. Une telle relation permet, par exemple, d'exprimer que l'on souhaite déployer l'application sur deux machines du groupe cible, peu importe lesquelles. Dans ce cas, des stratégies à appliquer lors de l'exécution du plan peuvent accorder une priorité aux machines offrant les meilleurs performances, ou à celles étant le moins surchargées, etc. L'ajout de tels opérateurs conduit à la définition d'un langage de description de plan de déploiement, à base d'opérateurs d'ordonnancement (comme *next*), d'opérateurs ensemblistes (*2parmi*) et d'opérateurs de composition (*and*).

Un statut peut aussi être associé à un plan de déploiement et permet d'exprimer dans quel état il se trouve. Les statuts suivants peuvent être identifiés :

- **INACTIF** : Le plan de déploiement n'a pas encore été exécuté.
- **ACTIF** : Le plan de déploiement est en cours d'exécution.
- **OK** : Le plan de déploiement a été réalisé avec succès.
- **KO** : L'exécution du plan de déploiement a échoué.

Un statut de résolution d'erreur pourrait aussi être ajouté. Dans ce cas, cela signifie que le plan de déploiement a commencé son exécution et qu'une erreur s'est produite. Dans le cas où des instructions de résolution d'erreur (activités de compensation ou de retour-arrière, par exemple) sont disponibles, elles peuvent être exécutées pour tenter de résoudre le problème. Ces instructions sont disponibles sous la forme d'un plan unitaire qui est alors relié par une relation « en cas d'erreur » avec le plan de déploiement de l'unité.

Pour assister le déploiement, cette étape de construction de plan de déploiement doit être automatique. La génération du plan global prend alors l'ensemble de l'information que nous avons défini.

6.3. Génération d'un plan de déploiement

C'est un administrateur qui demande la réalisation d'une activité de déploiement d'une (ou plusieurs) application(s) (liste(s) d'unités représentant chacune une version de l'application) sur un groupe de machines. Pour générer automatiquement un plan de déploiement pour une telle demande, il faut donc prendre en compte :

- le groupe *g* représentant le groupe sur lequel déployer l'application,
- le nom de l'application *n* à déployer : dans le cas où plusieurs unités (plusieurs versions d'une application) sont disponibles, il faut en sélectionner une (éventuellement une pour chaque machine du groupe),
- le nom de l'activité de déploiement *a* à réaliser.

Le plan de déploiement rassemble donc un ensemble de concepts du méta-modèle de déploiement. Il s'agit de la partie centrale du déploiement, puisque tous les concepts (entreprise, machine, unité) décrits précédemment sont utilisés pour le construire.

Les principales étapes du calcul d'un plan de déploiement permettant de déployer une unité sur un groupe de machines sont les suivantes :

- Sélectionner l'(les)unité(s) *u*, adaptée(s) à la (aux) machine(s) *m* cible(s) et respectant les stratégies de sélection du groupe *g* et de ses machines.
- Ordonner l'ensemble des plans unitaires (plan pour déployer une unité sur une machine) en fonction des stratégies du groupe pour former le plan de déploiement global. Des plans unitaires permettent de déployer l'unité, mais aussi chacune des dépendances. Ils peuvent être instanciés (si nécessaire) avec les caractéristiques de la machine. Chaque plan unitaire est considéré comme une brique de base du plan de déploiement global.
- Ajouter les cas d'erreur en respectant les stratégies.

La construction du plan de déploiement global implique donc l'application de diverses stratégies, que nous déterminerons dans la suite du document (Chapitre 6), à des moments différents.

7. EXEMPLE DE MODELISATION DE DONNEES DE DEPLOIEMENT

Dans cette partie, il s'agit de présenter un exemple des données qui viennent d'être décrites. La Figure 4.12 présente une partie du modèle d'entreprise de notre laboratoire de recherche : le *LSR*.

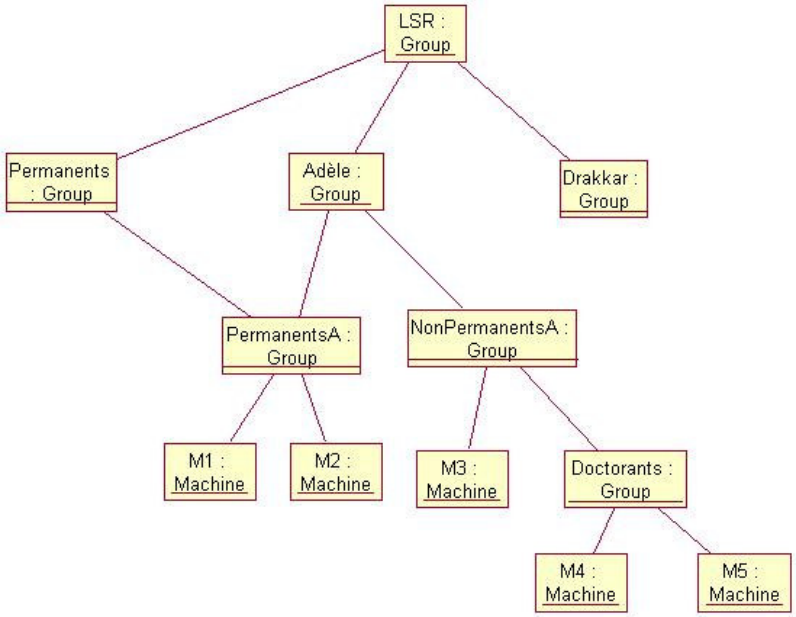

Figure 4.12 Description de l'entreprise *LSR*

Le groupe *LSR* est divisé en plusieurs sous-groupes. Certains concernent les équipes de recherche (ici, *Adèle* et *Drakkar*), d'autres concernent les utilisateurs (ici, *Permanents*). Le groupe Adèle est lui même divisé en plusieurs sous-groupes (*PermanentsA* et *NonPermanentsA*), qui peuvent être divisés en d'autres sous-groupes (*Doctorants*) ou contenir des machines (*M1*, *M2*, *M3*, *M4* et *M5*). On peut aussi noter que le groupe *PermanentsA* est aussi un sous-groupe du groupe *Permanents*. Remarquons que la structure du modèle est un graphe acyclique.

Chacune des machines citées a ses propres caractéristiques. La Figure 4.13 représente, par exemple, les caractéristiques de la machine *M1*. Les modèles d'instance des autres machines ne seront pas présentés ici, puisqu'ils sont similaires (seules les valeurs des attributs sont modifiées).

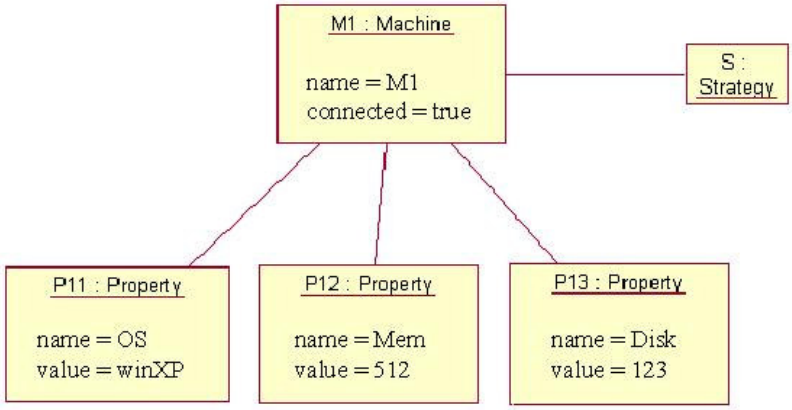

Figure 4.13 Description de la machine *M1*

La machine *M1* est caractérisée par trois propriétés *P11*, *P12* et *P13*. Elles concernent respectivement le système d'exploitation, la capacité mémoire et l'espace disque disponible sur la machine. De plus, le déploiement sur *M1* suit une stratégie *S*. La stratégie n'est pas détaillée ici, puisqu'un chapitre complet (Chapitre 6) traite des stratégies de déploiement.

Enfin, des unités de déploiement (versions d'une application) peuvent aussi être décrites. La Figure 4.14 décrit l'unité *U5*, qui est la version *5* de l'application *U*. Cette unité est décrite par trois propriétés (*P1-U5*, *P2-U5* et *P3-U5*), qui indiquent le type de l'unité (un *outil de développement*), son langage d'implémentation (*java*) et le type de son interface (*graphique*).

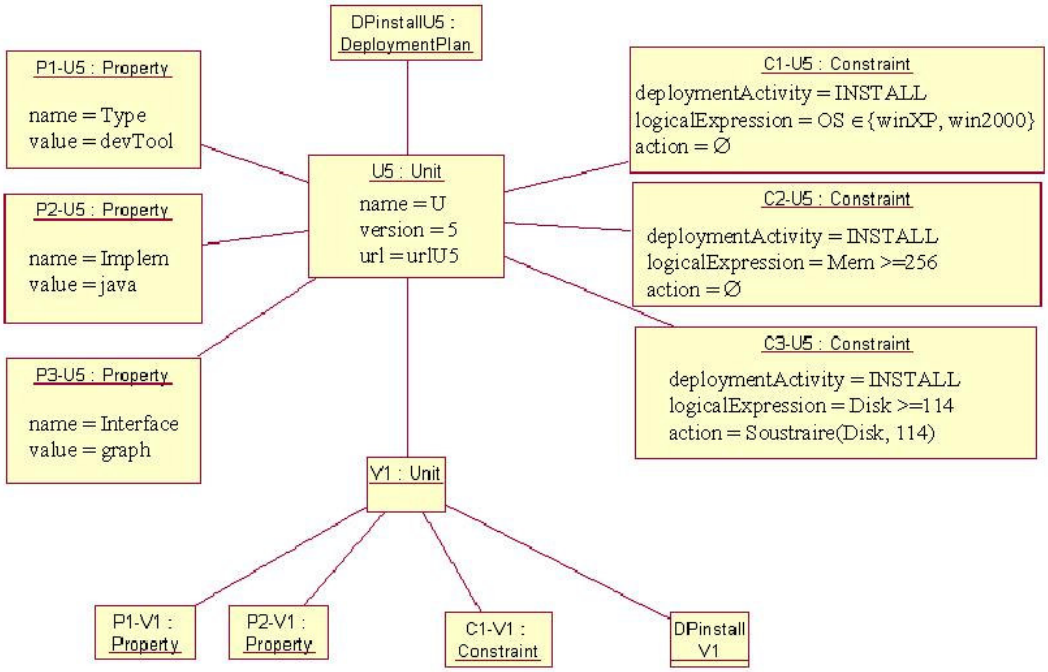

Figure 4.14 Description de l'unité de déploiement *U5*

De plus, des contraintes (*C1-U5*, *C2-U5* et *C3-U5*) imposent un système d'exploitation précis (*WindowsXP* ou *Windows2000*), une capacité mémoire minimum (supérieure ou égale à *256 Mo*) et un espace disque minimal (*114 Mo*) qui sera utilisé pour déployer l'unité. L'unité *U5* est aussi associée à un plan de déploiement (*DpinstallU5*) décrivant les étapes à réaliser pour l'installer sur une machine vérifiant les contraintes.

Enfin, l'unité *U5* a une dépendance vers l'unité *V1*. Cette unité a ses propres caractéristiques (*P1-V1*, *P2-V1*, *C1-V1* et *DPinstall3-U5*), dont les valeurs ne sont pas plus précisées ici.

8. SYNTHESE

Dans ce chapitre, nous avons précisé l'ensemble des concepts nécessaires au déploiement. Ces concepts constituent le méta-modèle de déploiement. Trois catégories de concepts principaux apparaissent : ceux liés à l'entreprise, ceux liés aux machines cibles, et ceux liées aux unités de déploiement. A partir de l'ensemble de ces concepts, un plan de déploiement peut être généré pour indiquer les étapes à réaliser pour déployer *n* unités de déploiement sur

p machines. Ce plan de déploiement est le résultat de l'application des contraintes et stratégies de déploiement et reflète les différents degrés de liberté qu'il reste pour son exécution.

A ces concepts sont associés des concepts complémentaires :

- Les **propriétés** décrivent les unités de déploiement et les machines cibles.
- Les **dépendances** entre unités indiquent de quelles autres unités a besoin une unité donnée.
- Les **contraintes** sont associées aux unités de déploiement. Pour chaque unité, elles indiquent quelles caractéristiques doit avoir une machine cible pour que l'unité puisse être déployée et fonctionner correctement.
- Les **stratégies** sont associées aux entités de l'entreprise (groupe et machine). Elles permettent de construire le plan de déploiement.

L'ensemble de ces concepts génériques peut ensuite être personnalisé pour mettre en œuvre une solution spécifique. Les concepts génériques sont alors associés à des concepts spécifiques, comme un modèle à composants, un modèle d'entreprise, etc. La mise en œuvre d'une solution est présentée dans le chapitre suivant.

CHAPITRE 5 : ORYA : MISE EN ŒUVRE DES MODELES ET CONCEPTION

Préambule :

Dans notre approche par les modèles, nous avons choisi d'utiliser des technologies étudiées dans notre équipe de recherche : les fédérations et les procédés. Ce chapitre présente ces technologies et les utilise pour mettre en œuvre les concepts que nous avons définis dans le chapitre précédent. Il décrit l'ensemble du travail de conception à réaliser pour fournir un environnement de déploiement.

---

1. INTRODUCTION

Deux technologies sont développées et expérimentées au sein de notre équipe de recherche : les procédés et les fédérations. C'est pourquoi nous les avons étudiées. La technologie des procédés [DAW98] (assimilable dans un premier temps à la technologie des workflows) permet de structurer les différentes étapes à réaliser pour atteindre un objectif. La technologie des fédérations, quant à elle, propose une solution, basée sur les techniques d'ingénierie dirigée par les modèles [Fav04], pour concevoir des applications à partir de méta-modèles.

Nous avons ensuite évalué dans quelles mesures ces technologies sont intéressantes pour définir un environnement de déploiement. Ceci nous conduit à définir une architecture du système de déploiement, basé sur les technologies des procédés et des fédérations.

Ce chapitre présente tout d'abord les technologies des procédés et des fédérations. Puis l'architecture du système de déploiement est décrite. Une partie indique aussi comment les concepts que nous avons définis dans le chapitre précédent sont utilisés, au travers de diverses structures. Enfin, le chapitre se termine par un exemple illustrant l'ensemble des informations expliquées.

2. LA TECHNOLOGIE DES PROCEDES

Un procédé représente la manière de réaliser un objectif. Plus qu'une automatisation des étapes à effectuer, il cherche à guider un utilisateur dans son travail. La maîtrise des procédés entraîne le contrôle de la qualité, des coûts et des délais, lors de la conception d'un logiciel.

La technologie des procédés a déjà été utilisée dans le cadre de la production d'applications. Elle permet notamment de modéliser, d'analyser, puis d'automatiser la production [DAW98]. Dans cette partie, il s'agit d'étudier dans quelle mesure ces concepts peuvent être appliqués ou adaptés au déploiement d'applications à grande échelle.

2.1. Modélisation de procédé

Une approche dirigée par des procédés permet de modéliser un but, et, de guider et d'automatiser sa réalisation. Ainsi, un procédé définit les activités et les entités qui les réalisent. Ces entités peuvent être des outils ou des entités humaines agissant pour effectuer une tâche.

La Figure 5.1 modélise les concepts de base des procédés [Der94]. Un procédé est vu comme un ensemble d'activités à réaliser. Une activité peut être composite, c'est-à-dire constituée d'un ensemble de sous-activités.

Un produit est une ressource nécessaire à la réalisation d'une activité. Une activité (*Activity*) reçoit un ensemble de produits (*Product*) en entrée. Ces produits peuvent être utilisés, supprimés ou modifiés au cours de la réalisation de l'activité. Un autre ensemble de produits représente alors le résultat de l'activité. Enfin, l'activité peut aussi utiliser des produits intermédiaires, pour réaliser des sous-activités ou étapes.

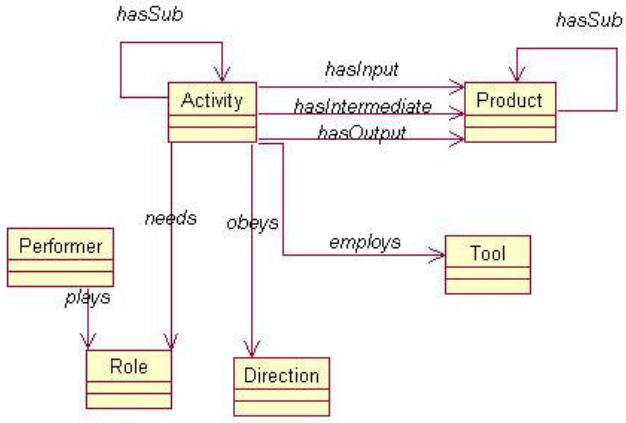

Figure 5.1 Concepts de base des procédés

Des agents (*Performer*), humains ou automatiques, sont chargés de réaliser les activités. Ils sont caractérisés par le rôle (*Role*) qu'ils remplissent. Les activités peuvent aussi être réalisées grâce à l'intervention d'outils (*Tool*).

Différentes vues d'un procédé peuvent être modélisées, selon les besoins et les acteurs concernés. Divers langages et environnements de procédés permettent de travailler avec ces concepts.

2.2. Langages et environnements de procédés

Les environnements de travail [BEM95] pour les procédés, appelés PSEE (*Process-sensitive Software Engineering Environment*), fournissent un support pour la technologie des procédés. Ils permettent de les définir, les modéliser et les exécuter. Un moteur de procédés contrôle l'environnement en interprétant le(s) langage(s) utilisé(s). Ainsi, il gère les flux de données qui sont échangés par les agents réalisant les activités.

Un environnement de travail pour les procédés permet aussi de stocker les informations et produits échangés par les différents acteurs intervenant dans un procédé. Les produits peuvent être partagés entre plusieurs acteurs ; mais, chaque acteur utilise son propre espace de travail pour réaliser l'activité dont il est responsable.

La Figure 5.2 schématise l'architecture d'un environnement de travail pour les procédés. D'une part, les procédés sont décrits dans un langage de modélisation de procédés. D'autre part, ils sont ensuite interprétés par le moteur de procédés qui charge des acteurs (machines ou humains) de le réaliser.

Les environnements de procédés utilisent des langages de modélisation de procédés, appelés PML (*Process Modelling Languages*), pour décrire les procédés. Certains de ces langages sont plus appropriés à l'une ou l'autre des étapes du procédé [CL95]. Par exemple, lors de l'exécution du procédé, le langage doit être opérationnel, tandis qu'il doit être compréhensible lors de sa phase de conception par un humain.

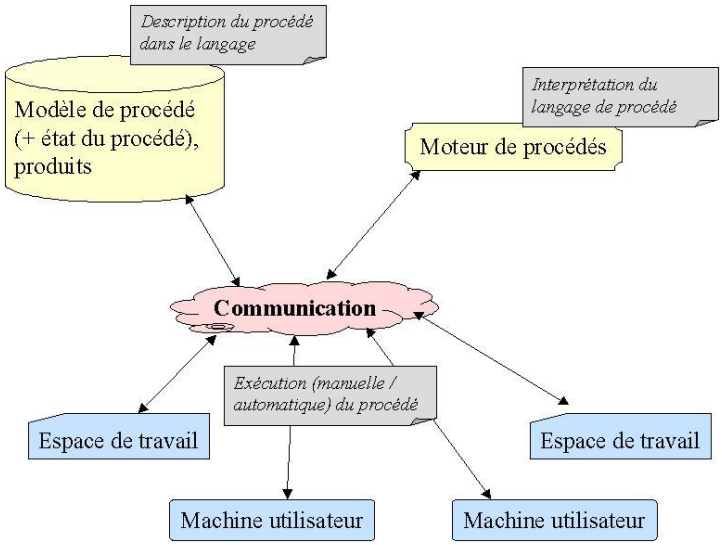

Figure 5.2 Environnement de travail pour les précédés

Plusieurs approches existent donc pour la définition des langages de modélisation de procédés. Par exemple, le langage Adele [BEM94] est basé sur une approche réactive. Il est basé sur l'utilisation de déclencheurs (*triggers*) et fournit des instances de procédés exécutables. Grâce aux langages de modélisation de procédés et aux environnement de travail qui les utilisent, il est possible de décrire et exécuter un procédé.

APEL [DEA98, EVL+03], développé dans notre équipe de recherche, fournit un formalisme graphique de haut niveau pour la modélisation des procédés. Cette approche gère aussi l'exécution de ces mêmes procédés, via un moteur de procédés. Un système d'agenda permet aussi à chaque agent humain de visionner l'ensemble de l'exécution du procédé et de savoir quand il doit intervenir. Le paragraphe suivant détermine quels services ces technologies peuvent apporter au déploiement

2.3. La technologie des procédés au service du déploiement

L'utilisation de la technologie des procédés permet essentiellement d'automatiser le déploiement et de contrôler son déroulement. Elle peut aussi être utilisée pour automatiser la gestion des erreurs qui peuvent survenir.

2.3.1. Automatisation du déploiement

Le déploiement est composé de plusieurs étapes, comme la sélection d'une configuration adaptée au client ou le transfert de données. Chaque activité du cycle de vie du déploiement peut aussi comporter plusieurs étapes, comme la copie de fichiers ou la création de variables d'environnements. Chacune de ces étapes est nécessaire au bon déroulement du déploiement complet d'un produit chez un client. Les procédés sont une bonne technique pour ordonner et structurer l'ensemble de ces étapes et des données utilisées.

Avec un procédé exécutable, il est aussi plus facile d'automatiser le déploiement, même si une intervention humaine reste possible à tout instant. L'automatisation du déploiement permet ainsi de déployer un produit sur un grand nombre de machines, avec un minimum d'intervention humaine.

De plus, l'utilisation de procédés offre la possibilité de fournir une séquence d'activités reproductible. En effet, dans le cadre du déploiement à grande échelle, il est important de pouvoir exécuter plusieurs fois la même séquence d'instructions, pour, par exemple, pouvoir déployer une application sur plusieurs cibles ou prendre en compte des erreurs. De même il est intéressant de pouvoir définir une séquence « générique » afin de pouvoir la réutiliser en l'adaptant en fonction de situations particulières

Nous avons choisi d'utiliser APEL [DEA98], un environnement de support de procédés développé par notre équipe de recherche. Cet environnement permet de représenter, sous forme graphique, les activités, les flux de contrôle (*workflow*) et de données (*dataflow*) du procédé de déploiement.

Les activités ainsi décrites seront réalisées au cours de l'exécution du procédé. Ainsi, les tâches seront affectées les unes après les autres à un utilisateur (via un système d'agenda) ou à un outil. Quand une tâche est terminée, on passe à celle qui lui succède dans le descriptif de l'activité, et ainsi de suite jusqu'à ce que l'activité soit entièrement terminée.

2.3.2. Gestion des erreurs

Des procédés de déploiement peuvent être définis, puis exécutés. Cependant, des erreurs surviennent parfois au cours du déploiement. Elles doivent être repérées afin de laisser le(s) site(s) client(s) concerné(s) dans un état cohérent. Un travail [Mar04] a été réalisé afin d'étudier l'aspect transactionnel du déploiement en identifiant les cas d'erreur et leurs alternatives possibles.

2.3.2.1. Aspect transactionnel d'un procédé de déploiement

Pour un procédé de déploiement, il est intéressant de repérer quoi faire en cas d'erreur au cours de l'activité. Un choix peut être de revenir à l'état initial en défaisant les activités déjà réalisées. Une autre alternative peut être de tenter une activité équivalente pour remplacer celle où l'erreur s'est produite. Une décision peut aussi être prise de revenir à un état cohérent du site cible, sans défaire toutes les activités déjà effectuées. Ce dernier choix permet ensuite de reprendre le déploiement là où il a été stoppé.

Différentes stratégies peuvent ainsi être appliquées en cas d'erreur. Il est alors important d'étudier l'aspect transactionnel d'un procédé de déploiement. Les modèles de transaction avancées tentent de résoudre les problèmes liés à une exécution complexe [Elm90], comme un procédé de déploiement, et permettent ainsi de définir plusieurs types d'activités [MBD04].

2.3.2.2. Les types d'activités d'un procédé

La Figure 5.3 illustre les différents concepts et types d'activités qui peuvent être utilisés pour gérer les erreurs lors du déploiement :

- Une **activité de compensation** permet de revenir en arrière en cas d'erreur. Ainsi, si une erreur survient au cours d'une activité du procédé de déploiement, une activité de compensation peut défaire les opérations réalisées par l'activité où a eu lieu l'erreur, et permettre de revenir à un état cohérent pour le site client.

- Les **activités non-critiques** sont des activités qui n'empêchent pas le bon déroulement du déploiement. Ainsi, si une erreur survient au cours de l'exécution d'une activité non-critique, le procédé poursuivra quand même son exécution avec les activités suivantes.
- Des **points de reprise** peuvent aussi être définis. Ils permettent de repérer les instants où le procédé peut être interrompu. Ce sont des points où le site client est dans un état cohérent. Si une erreur se produit au cours de l'exécution du procédé de déploiement, plutôt que de défaire toutes les étapes réalisées, seules les étapes permettant d'atteindre un point de reprise seront défaites. Le déploiement pourra ensuite reprendre à partir de ce point de reprise.
- Des **activités de contingence** peuvent aussi être définies. Elles permettent de réaliser une activité d'une autre manière. Par exemple, l'activité de transfert peut être remplacée par l'activité de contingence correspondante. Dans ce cas, les deux activités réalisent un transfert qui utilisent, par exemple, deux protocoles différents. Dans ce cas, si l'activité échoue, son activité de contingence prend le relais pour terminer l'opération.

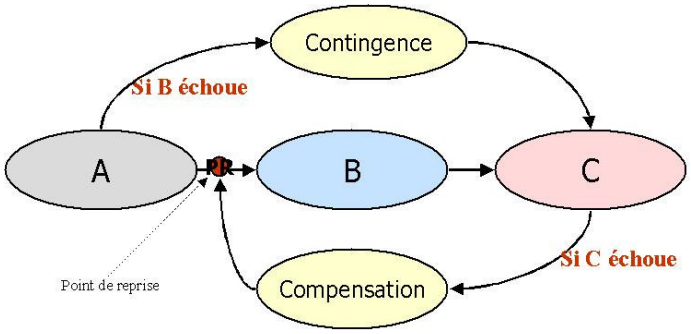

Figure 5.3 Transaction de déploiement

Ces différents types d'activités permettent de faciliter le traitement des erreurs lors du déploiement. Ainsi, si des activités de contingence ou de compensation sont définies pour chaque activités du plan (procédé) de déploiement, il est alors possible de tenter de résoudre des erreurs qui peuvent survenir, en utilisant ces activités « secondaires » et sans demander l'intervention d'un utilisateur.

La technologie des procédés semble donc bien adaptée pour automatiser le procédé de déploiement, de contrôler son bon déroulement et gérer les erreurs qui peuvent se produire. L'apport principal des procédés est l'automatisation du traitement, qui ne demande aucune intervention de l'utilisateur.

3. LA TECHNOLOGIE DES FEDERATIONS

L'interopérabilité est la capacité qu'ont certains outils hétérogènes de fonctionner ensemble et de donner un accès à leurs ressources. Plus formellement, deux utilisateurs *A* et *B* inter-opèrent si *A* demande un service à *B*, que *B* le comprend et peut le résoudre (et inversement). Ce concept implique donc l'utilisation de concepts communs entre les différents utilisateurs impliqués.

Les fédérations d'outils ont été initialement conçues pour proposer une solution pour résoudre le problème d'interopérabilité [ECB98]. Elles se sont ensuite orientées vers une approche IDM (Ingénierie dirigée par les modèles). L'idée est de construire des domaines autonomes et exécutables [IEV05]. Dans ces domaines, les concepts (méta-modèle) sont séparés de l'implémentation. Ainsi, les outils ou composants peuvent être choisis en fonction des préférences du client ou de tout autre critère. Ces domaines peuvent ensuite être « composés » en établissant des relations entre leurs méta-modèles.

L'approche des fédérations est la composition d'éléments logiciels de diverses natures entre eux. La composition est basée sur la coordination et dirigée par la composition de modèles exécutables [Vil03]. Les premiers objectifs sont de :

- préserver l'autonomie et l'initiative des participants (éléments logiciels),
- faciliter l'évolution de l'application globale (résultante de la composition).

3.1. Domaine

L'approche par fédération de domaines permet d'obtenir des spécifications de haut niveau exécutables et de créer une bibliothèque de modèles et méta-modèles. Ces modèles et méta-modèles peuvent ensuite être facilement intégrés, réutilisés et adaptés sans avoir à les modifier [EV05].

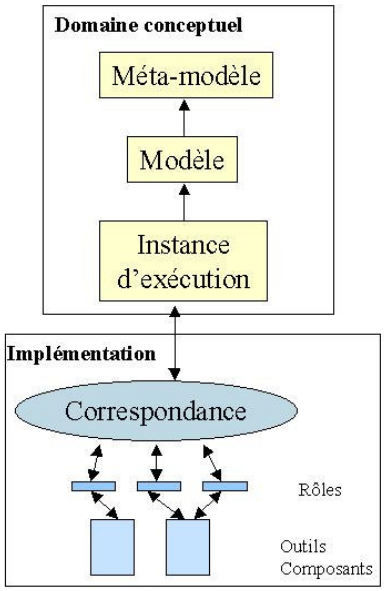

Figure 5.4 Niveaux conceptuels et niveau d'implémentation d'un domaine

La Figure 5.4 indique qu'un domaine est composé de deux parties [LEV03] :

- Une **partie abstraite** (*Domaine conceptuel*) représente le modèle conceptuel du domaine et contient la sémantique abstraite.
- Une **partie concrète** (*Implémentation*) représente la réalisation du domaine : elle est constituée d'un ensemble de participants qui concrétisent les concepts abstraits. Des participants différents peuvent être utilisés pour réaliser un

domaine, dès qu'il sont capables de concrétiser la sémantique abstraite du modèle.

Un domaine abstrait est constitué de trois niveaux (Figure 5.4) :

- le **méta-modèle** (modèle du domaine) qui contient les concepts communs au domaine,
- le **modèle** spécifique à l'application (conforme au méta-modèle), et,
- les **instances**.

Le concept de domaine permet d'isoler un groupe de participants. Les participants d'un domaine se coordonnent en utilisant l'architecture d'une fédération pour fournir un ensemble de services. On appelle cela la composition verticale car elle associe les concepts abstraits d'un domaine avec les implémentations concrètes de ces concepts (outils, code, …)

Pour créer une application plus grande et plus complexe, les domaines peuvent être composés entre eux [Veg05]. Pour cela, il faut établir les liens entre leurs méta-modèles. Ces liens comprennent les associations entre concepts différents et les correspondances entre concepts similaires.

3.2. Composition de domaines

Lorsqu'une relation entre les concepts de deux domaines existe à un haut niveau d'abstraction, la composition de domaines permet de remplacer les connexions directes entre les participants (niveau implémentation) par des connexions entre les concepts des modèles conceptuels des deux domaines (Figure 5.5). Ainsi, lorsqu'un utilisateur veut créer une nouvelle application avec des concepts déjà existants, il peut réutiliser ses concepts en composant des domaines [EIV05], sans toucher aux éléments logiciels (participants) spécifiques à chaque domaine.

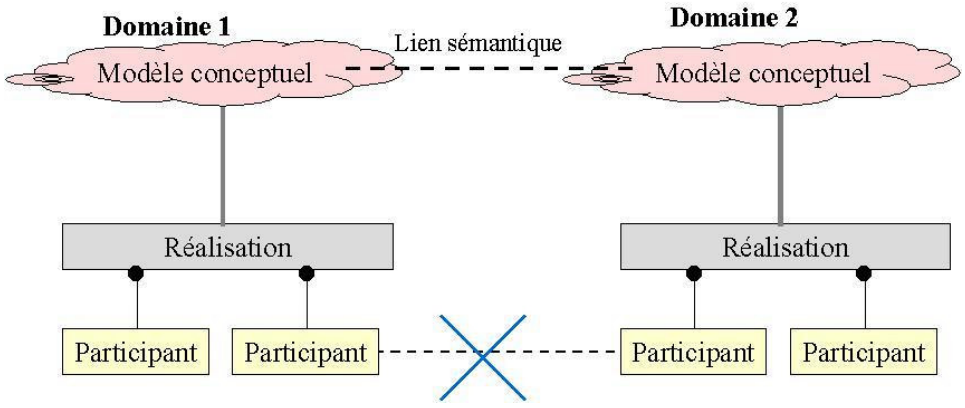

Figure 5.5 Lien entre domaines

Par exemple, pour une application de gestion documentaire, les domaines de procédé et de document seront composés pour créer le domaine de l'application globale.

De plus, deux types de composition peuvent exister :

- Une **composition horizontale** lie deux concepts de haut niveau, issus des domaines conceptuels (« Lien sémantique » de la Figure 5.5).
- Une **composition verticale** permet de lier un concept de haut niveau à la manière de le réaliser (liaison entre un domaine conceptuel et sa réalisation).

3.3. Niveau concret : les concepts de base

Les concepts communs constituent l'univers commun [EVC01-b], qui permet aux éléments logiciels de se synchroniser (données et état).

L'**univers commun** matérialise les concepts partagés entre les participants d'une manière qu'ils comprennent tous. La coordination (cohérence des concepts) entre les participants est traduite par la synchronisation de l'état de leurs instances avec l'instance du concept partagé dans l'univers commun (et inversement).

Lorsqu'un élément logiciel (participant) existant est utilisé dans une fédération, il ne doit/peut généralement pas être modifié : un adaptateur permet alors de faire le lien entre les concepts communs (univers commun) et les concepts utilisés par le participant (composition verticale). De plus, pour permettre à différents participants d'inter-opérer entre eux sans se connaître, le concept de rôle est introduit. Un **rôle** représente une abstraction des fonctionnalités d'un participant. Ainsi, lorsque cela est nécessaire, un participant peut faire appel à un autre par l'intermédiaire de son rôle.

Un rôle a :

- une **interface** qui regroupe les méthodes fournies par le rôle,
- des **droits** qui lui permettent d'appeler des méthodes de l'univers commun et d'écouter les événements qui sont émis.
- la possibilité d'émettre des **événements**.

Plusieurs participants peuvent remplir le même rôle, c'est-à-dire que plusieurs outils peuvent fournir les mêmes services. Dans le cas où plusieurs outils sont disponibles, un seul est choisi pour être utilisé. Un participant peut aussi remplir plusieurs rôles, c'est-à-dire fournir plusieurs services.

3.4. La technologie des fédérations au service du déploiement

Plusieurs domaines sont utilisés dans le déploiement : les concepts sont liés à l'entreprise, aux sites cibles, aux unités de déploiement et au plan de déploiement. Il semble donc intéressant de définir une fédération pour permettre à ces différents participants de coopérer pour réaliser entièrement la tâche du déploiement.

Ces différents domaines sont modélisés de manière générique (par rapport au déploiement) dans le méta-modèle de déploiement que nous avons défini dans le chapitre précédent. Les différentes partie du méta-modèle peuvent être réalisées par des modèles différents. Ainsi, par exemple, la partie liée aux unités de déploiement peut être remplie par un modèle à composants spécifique. En utilisant la composition de domaines, ces concepts pourront être réutilisés dans le domaine du déploiement en définissant les liens qui existent entre les concepts déjà définis et les concepts utilisés dans le déploiement. Il est donc particulièrement

intéressant de composer les différents domaines concernés pour créer l'environnement global de déploiement. Cette caractéristique permet ainsi la réutilisation de concepts mais aussi de données et outils déjà existants.

3.5. L'association des procédés et des fédérations

Il est intéressant d'associer la technologie des procédés à une fédération [EVL+03] pour le déploiement. En composant un domaine de procédés avec les concepts de déploiement, il est plus simple de gérer les étapes à réaliser pour déployer une application. En effet, un procédé de déploiement peut être défini pour structurer les différentes étapes à réaliser (transfert, résolution de dépendances, exécution de fichiers de commandes, …) pour remplir la tâche de déploiement. Avec une telle utilisation des technologies de fédérations et de procédés, il est alors relativement aisé d'automatiser le déploiement.

De plus, en se plaçant à un niveau plus bas, la fédération, grâce au concept de rôle, fournit une grande flexibilité à l'utilisateur. En effet, elle permet d'utiliser des outils différents d'une machine à l'autre. Par exemple, il est possible d'utiliser un outil de transfert sur un site client, et un autre outil de transfert, n'utilisant pas le même protocole, sur un autre site client. Comme les outils sont utilisés par la fédération via leurs rôles (ici, rôle de transfert), le déploiement peut être réalisé dans les deux cas sans modification du procédé.

Cette approche est aussi extensible, par le nombre illimité de machines qu'il est possible de gérer. De plus, les machines ne se connaissent pas entre elles. Il est ainsi possible de déployer une application sur un site client sans connaître quel serveur d'applications la fournie : c'est la fédération qui se charge de faire la recherche.

La technologie des fédérations, alliée à la technologie des procédés, semble donc une solution intéressante pour résoudre la problématique du déploiement. Pour vérifier ces avantages, nous définissons l'architecture de notre système de déploiement, en utilisant ces deux technologies.

4. ARCHITECTURE DU SYSTEME

Le déploiement met en jeu des informations liées (Figure 5.6) :

- aux unités qui sont déployées,
- aux machines qui reçoivent les unités,
- à l'entreprise où sont déployées les unités, et,
- au plan de déploiement utilisé pour déployer les unités.

Notre environnement de déploiement, ORYA (Open enviRonment to deploY Applications), est basé sur les technologies des fédérations et des procédés. Cette partie décrit l'architecture de l'environnement sur la base de ces technologies.

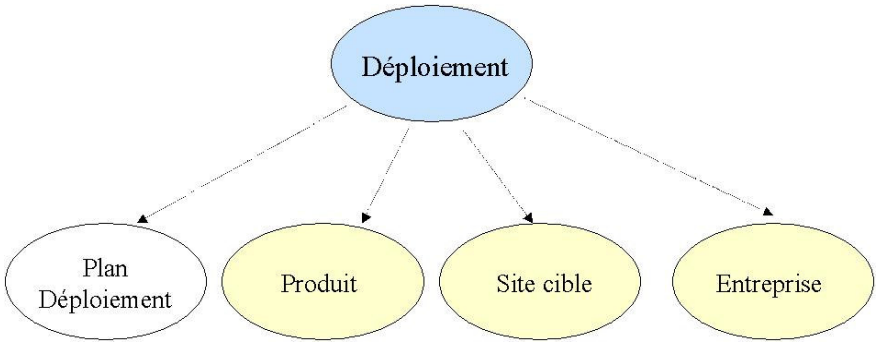

Figure 5.6 Informations liées au déploiement

4.1. Obtention des différents modèles

Dans le chapitre précédent, nous avons séparé le méta-modèle de déploiement en trois parties principales, et une quatrième obtenue à partir des autres (plan de déploiement). Ces trois parties constituent les modèles des différents domaines conceptuels qui sont nécessaires au déploiement. Chacun de ces modèles est obtenu d'une manière différente.

La quatrième partie, celle concernant le plan de déploiement, est celle que nous produisons à partir des informations des autres domaines, et plus particulièrement en appliquant des stratégies de déploiement.

Le **modèle de produit**, dont le concept principal que nous aurons à manipuler est celui d'unité de déploiement, est obtenu lors du *packaging* de l'application et de sa mise à disposition sur un serveur d'applications. Le modèle de produit est fourni par le producteur. Il peut être fourni sous une forme complexe, mais contient l'ensemble de l'information nécessaire pour construire le modèle de produit utilisé dans le contexte du déploiement. A ce moment là est aussi fourni le plan de déploiement unitaire indiquant le procédé de déploiement de l'unité. Le **plan de déploiement** global, qui sera ensuite calculé lorsqu'un déploiement sera demandé au niveau de l'entreprise, utilisera le plan de déploiement unitaire de l'unité.

Dans le **modèle de site cible**, le concept principal est celui de machine. Ce modèle est obtenu à partir du modèle de site qui décrit chaque machine. Il peut lui aussi être très complexe, mais contient au moins l'information nécessaire. Ce modèle peut aussi être obtenu par introspection de la machine, si des outils sont disponibles pour réaliser cette opération. Il faut rappeler, que même si nous avons choisi le terme de « machine », une cible peut être un PC, un ordinateur portable ou même une grappe. Dans ce dernier cas, la grappe est considérée comme une cible unitaire pour le déploiement, même si d'autres opérations seront éventuellement réalisées localement.

Enfin, le **modèle d'entreprise** a pour concept principal la notion de groupe. Il est obtenu à partir du modèle d'entreprise, déjà défini en termes d'équipes, de départements, d'utilisateurs, etc. Si aucun modèle d'entreprise n'est disponible, un modèle contenant l'information nécessaire au déploiement est défini. S'il est obtenu à partir d'un modèle existant, il peut s'agir d'une simple mise en correspondance de concepts (par exemple, un département est un groupe). Il peut aussi s'agir d'un « calcul » plus complexe. Par exemple, certaines cibles peuvent être supprimées du modèle initial : dans ce cas, elle ne feront pas partie des cibles de

déploiement. La structure hiérarchique (groupes et sous-groupes) de l'entreprise peut aussi être modifiée/adaptée pour le déploiement, etc.

4.2. Réutilisation de domaines issus des fédérations

Plusieurs domaines ont déjà été définis dans l'approche des fédérations. Ils sont donc disponibles pour être réutilisés. Ainsi, nous choisissons de réutiliser les domaines de produit (*Product Domain*) et d'activité (*Activity Domain*). Il ne reste alors qu'à définir les modèles conformes aux méta-modèles qui seront utilisés dans le cadre du déploiement.

Avec le **domaine de produit**, nous définissons le modèle de produit qui permet de décrire les unités de déploiement. Les propriétés et les contraintes peuvent alors être définis comme des attributs de l'unité.

Avec le **domaine d'activité**, nous définissons le modèle de plan de déploiement. Ainsi, un plan de déploiement global peut être vu comme un procédé, alors qu'un plan de déploiement unitaire est une activité (éventuellement composite) de ce procédé. Le domaine d'activité couvre donc les deux types de plans, mais avec des concepts différents.

A partir de ces domaines déjà existants, nous pouvons définir notre environnement de déploiement, construit comme une composition de plusieurs domaines.

4.3. Approche utilisant les technologies de fédérations et procédés

Les fédérations permettent tout d'abord de pouvoir composer différents domaines pour obtenir celui qui définit l'environnement de déploiement. Chacun des sous-domaines peut être réutilisé séparément ou pour créer d'autres domaines composites. L'approche par fédérations permet aussi, par transformation de modèles, de spécialiser chacun des sous-domaines, et plus particulièrement chacun des concepts, avec des modèles plus spécifiques.

Dans le chapitre précédent, nous avons vu que le méta-modèle de déploiement comporte des données relevant de thèmes différents. C'est pourquoi il est important de créer le domaine (au sens fédérations) de déploiement à partir de plusieurs autres domaines. Ainsi, comme l'indique la Figure 5.7, le domaine de déploiement est créé à partir de l'association [EIV05] des domaines d'entreprise, de site cible, de produit et d'activité (procédé). La composition de tous ces domaines entre eux forme le domaine de déploiement. Pour former le domaine de déploiement, il faut aussi ajouter un nouveau concept : celui de stratégie.

Le concept de stratégie est ainsi traité différemment, et indépendamment des données d'entreprise. En effet, les entreprises ne modélisent pas forcément des stratégies dans leurs données. Ainsi, le domaine d'entreprise concerne essentiellement la structure de l'entreprise (groupes et machines). Le concept de stratégie apparaît alors comme un concept émergeant du déploiement. Cette vision permet alors de faciliter la réutilisation de modèles d'entreprises (avec ou sans stratégies) déjà existants.

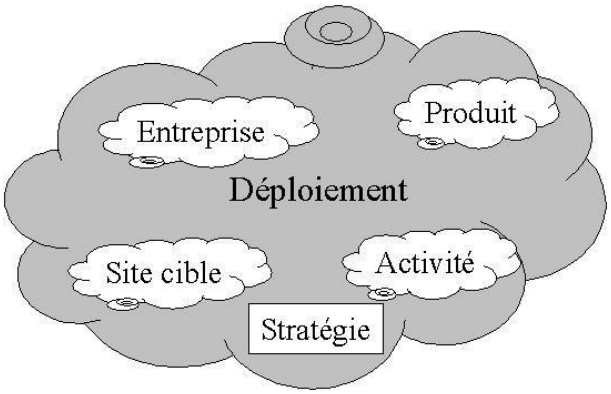

Figure 5.7 Domaine de déploiement

La Figure 5.8 illustre la composition entre les différents domaines que nous venons de citer. Elle indique aussi les différentes relations entre ces domaines.

La relation (correspondance) entre le domaine d'entreprise et de site cible (relation (1) de la Figure 5.8) se fait entre les concepts de machine. En effet, le domaine d'entreprise contient les concepts de groupe et machine. Celui de site cible permet de décrire une machine, avec ses propriétés et les unités qui y sont déployées.

La relation (correspondance) entre les domaines de site cible et de produit (relation (2) de la Figure 5.8) se fait au travers de la notion d'unité et de révision. Le domaine de produit permet de décrire un produit, avec ses différentes branches et révisions. Même si ces concepts ne sont pas utiles dans le domaine de déploiement, ce domaine peut permettre de gérer la partie « serveur d'applications ». En effet, dans le domaine de déploiement, seule la notion d'unité est utilisée, mais pour le producteur, il est intéressant de pouvoir gérer les différentes évolutions de son produit.

La relation (association UML) entre les domaines de produit et d'activité (relation (3) de la Figure 5.8) se fait au travers de la relation entre une révision et une activité. Le domaine d'activité permet de décrire les procédés. Il est utilisé pour représenter le plan de déploiement d'une unité de déploiement. La composition entre les domaines d'activité et de produit définit déjà le domaine d'APEL, qui implémente un moteur de procédé. Le domaine d'activité est en réalité plus complexe : une partie a été cachée pour simplifier la spécification que nous donnons ici. En effet, à ce niveau, il n'est pas nécessaire de connaître plus de détails. D'un point de vue sémantique, cette relation associe une unité de déploiement (*Revision*) au plan de déploiement unitaire (*Activity*) qui permet de la déployer.

La relation (association UML) entre les domaines d'entreprise et d'activité (relation (4) de la Figure 5.8) permet au domaine d'activité de représenter le plan de déploiement global. En effet, un plan de déploiement global est vu comme un procédé. Chacune des activités du procédé permet de déployer une unité sur une machine (plan de déploiement unitaire). Ces activités peuvent être composites : dans ce cas, elles représentent la suite des étapes à réaliser pour déployer l'unité. D'un point de vue sémantique, cette relation représente l'association qui est créée au moment d'un déploiement. Ainsi, un groupe (*Group*) est associé à un plan de déploiement global (*Process*), qui permet de déployer un ensemble d'unités sur les machines du groupe.

Enfin, **le concept de stratégie** est ajouté à l'association de tous ces domaines pour créer le domaine de déploiement et définir une extension du méta-modèle d'entreprise. Les stratégies sont alors associées aux concepts de l'entreprise, groupe et machine (relations (5) et (6) de la Figure 5.8). Elles permettront ensuite de créer les différents plans de déploiement globaux. Le concept de stratégie n'appartient pas au domaine d'entreprise. En effet, une entreprise est avant tout un ensemble de machines structurées, la notion de stratégie est plus ou moins indépendante de l'entreprise et plus particulièrement liée au déploiement au sein d'une entreprise.

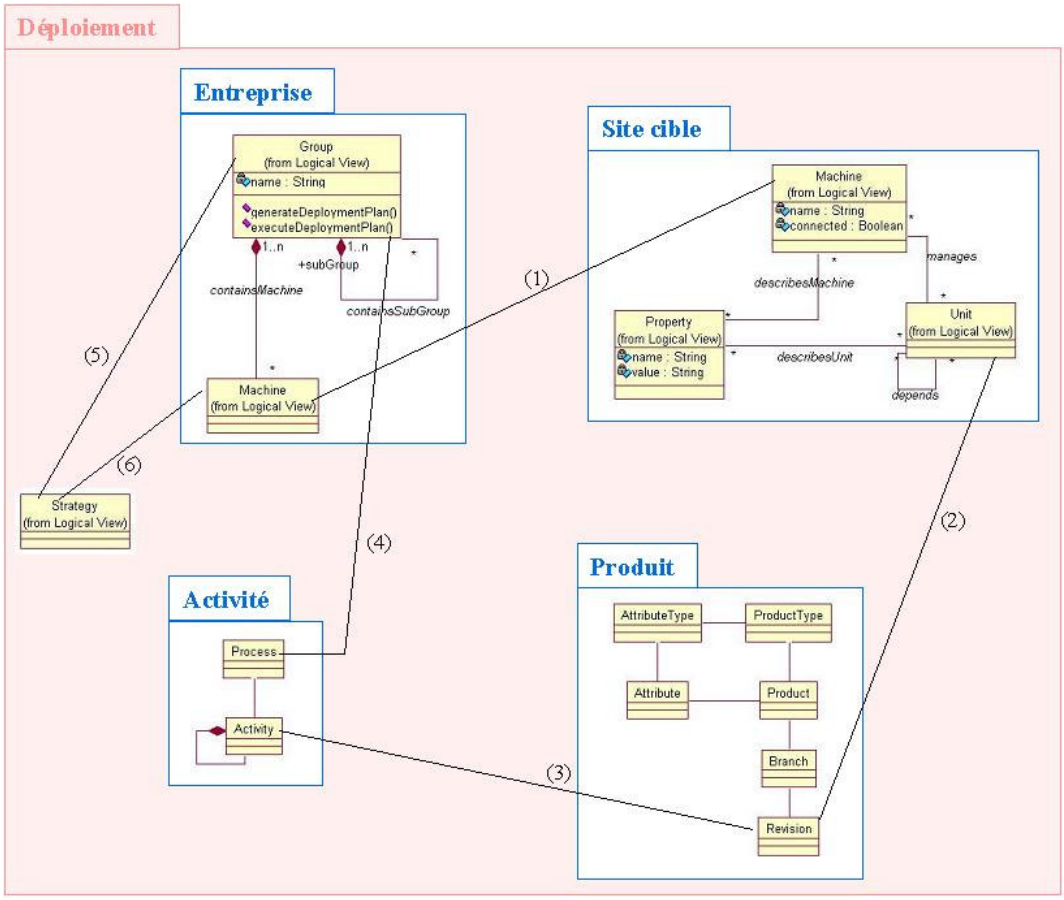

Figure 5.8 Composition de domaines

La Figure 5.8 représente le domaine de déploiement. Sur cette figure, nous pouvons aussi remarquer la différence de sémantique entre les associations utilisées [ES05, San05] :

- Une **extension du méta-modèle** permet d'ajouter des concepts. C'est le cas des relations entre le domaine d'entreprise et le concept de stratégie.
- Un **raffinement** permet d'étendre un élément (ou plusieurs) pour qu'il puisse contenir tous les détails qui le concerne : il s'agit du même concept qui est défini différemment (correspondance de concepts des deux domaines). C'est le cas des relations entre les domaines d'entreprise et de site cible, qui précise le concept de machine. C'est aussi le cas entre les domaines de site cible et de produit, qui précise le concept d'unité.

- Une **composition** permet de faire interopérer plusieurs environnements en liant les concepts du méta-modèle. C'est le cas des associations entre les domaines d'activité et d'entreprise, et entre les domaines d'activité et de produit qui définissent, respectivement, les concepts de plan de déploiement global et plan de déploiement unitaire comme un procédé ou une activité de procédé. La relation entre un groupe et un plan de déploiement est créée lorsque un plan de déploiement global est construit pour déployer un ensemble d'unités sur le groupe.

4.4. Spécification des modèles

Les différents types d'information des différents domaines peuvent être disponibles dans l'environnement où les unités sont déployées. C'est pourquoi, il est intéressant d'utiliser les fédérations pour établir la correspondance entre les différentes données déjà existantes et les concepts de déploiement auxquels elles se rapportent. Ainsi, par une série de transformations de modèles, les différents domaines peuvent être spécialisés/adaptés selon les besoins de l'entreprise.

Ainsi, par exemple, les informations sur la structure de l'entreprise utilisées pour le déploiement peuvent être obtenues à partir des données du modèle d'entreprise.

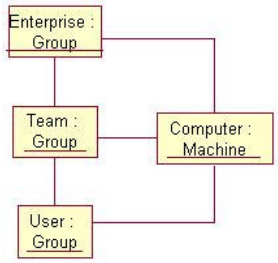

Figure 5.9 Modèle d'entreprise

La Figure 5.9 représente un exemple de modèle d'entreprise. Nous définissons les correspondances suivantes avec les concepts du méta-modèle de déploiement (Figure 4.3):

- Les concepts d'entreprise (*Enterprise*), d'équipe (*Team*) et d'utilisateur (*User*) du modèle d'entreprise appartiennent tous au type groupe du méta-modèle de déploiement. Les relations de composition sont conservées avec la hiérarchie de groupes et sous-groupes.
- Le concept d'ordinateur (*Computer*) du modèle d'entreprise est de type machine. Il faut cependant noter que la simple relation du modèle d'entreprise entre les concepts d'utilisateur et de machine est transformée en relation d'agrégation dans le méta-modèle de déploiement. En effet, pour le déploiement, l'information intéressante est qu'un groupe « utilisateur » contient un ensemble de machines, et non qu'un utilisateur utilise plusieurs machines.

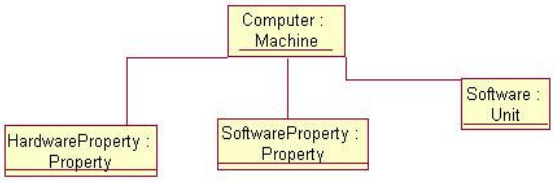

Figure 5.10 Modèle de site cible

La Figure 5.10 représente un exemple de modèle de site cible. Nous définissons les correspondances suivantes avec les concepts du méta-modèle de déploiement (Figure 4.6):

- Le concept d'ordinateur (*Computer*) est de type machine.
- Les concepts de propriétés matérielles (*HardwareProperty*) et logicielles (*SoftwareProperty*) sont de types propriétés (*Property*).
- Les logiciels (*Software*) sont des unités.

De la même manière, un modèle de produit doit être défini. Le modèle de produit peut définir des applications de tous types (COTS, applications à base de composants [OSK04, SOK04], …). Il suffit d'établir quels concepts du modèle sont conformes aux concepts du méta-modèle.

Il faut aussi noter que dans un modèle à base de composants, par exemple, tous les concepts ne seront pas forcément nécessaires au déploiement. Le modèle qui spécialise les concepts du déploiement peut alors être un sous-modèle du modèle à composants complet.

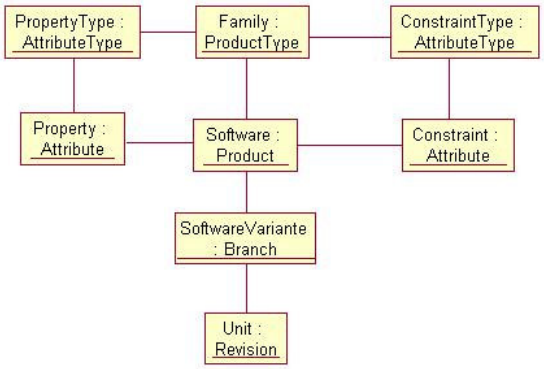

Figure 5.11 Modèle de produit

La Figure 5.11 représente un exemple de modèle de produit (pour COTS). Ainsi, nous définissons les correspondances suivantes avec les concepts du méta-modèle de produit (Figure 4.8) :

- Le concept d'unité (*Unit*) est de type révision (*Revision*).
- Les concepts de propriétés (*Property*) et de contraintes (*Constraint*) sont de type attribut (*Attribute*). Ils sont reliés à leurs types d'attribut (respectivement *PropertyType* et *ConstraintType*)

- Les logiciels (*Software*) sont des produits (*Product*), appartenant à une famille de logiciels (*Family*), comme les traitements de texte, etc. Ils sont divisés en plusieurs branches (*SoftwareVariante*).

Chaque partie du méta-modèle peut alors être spécialisée par des transformations de modèles. L'opération est à réaliser pour chaque sous-domaine (entreprise, site cible, produit, activité) et/ou concept (stratégie) du domaine de déploiement.

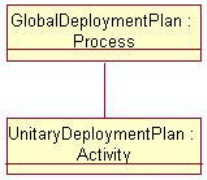

Figure 5.12 Modèle de plan de déploiement

La Figure 5.12 représente un modèle de plan de déploiement. Ainsi, nous définissons les correspondances suivantes avec les concepts du méta-modèle d'activité (Figure 5.8) :

- Le concept de plan de déploiement global (*GlobalDeploymentPlan*) est de type procédé (*Process*).
- Le concept de plan de déploiement unitaire (*UnitaryDeploymentPlan*) est de type activité (*Activity*).

Ensuite, des instances d'exécution peuvent être définies comme instances de ces modèles. Puis le niveau implémentation de l'approche fédérations (Figure 5.4) permet de définir l'univers commun et les différents rôles et outils nécessaires. Ce niveau sera présenté dans le chapitre sur la réalisation d'ORYA.

5. STRUCTURES DE DEPLOIEMENT

Pour réaliser les différentes étapes du déploiement, des structures de données servent à rassembler l'information qui est nécessaire, et qui est extraite des instances, issues des concepts du déploiement. Ainsi, nous définissons tout d'abord deux structures :

- La **structure d'entreprise** permet de représenter l'environnement cible.
- La **structure d'application** permet de rassembler les différentes unités de déploiement possibles.

Chacune de ces structures est associées à un déploiement précis d'un ensemble d'unités de déploiement sur *n* machines. Ainsi, la structure d'application indique quoi déployer, et la structure d'entreprise indique où déployer. L'étude de ces deux structures permet ensuite de construire la **structure de plan**, qui représente le plan de déploiement global.

5.1. Structure d'entreprise

La structure d'entreprise est basée sur un sous arbre du modèle d'entreprise complet. Elle représente l'ensemble des groupes et machines sur lesquels le déploiement se produit.

La Figure 5.13 montre comment la structure d'entreprise est déduite du modèle de l'entreprise. Ici, c'est un sous-arbre complet qui est concerné. Mais certains sous-groupes ou machines auraient pu être supprimés ou ajoutés. Ainsi, par exemple, le déploiement aurait pu ne pas concerner l'utilisateur *User1* ou concerner aussi la machine *M7*.

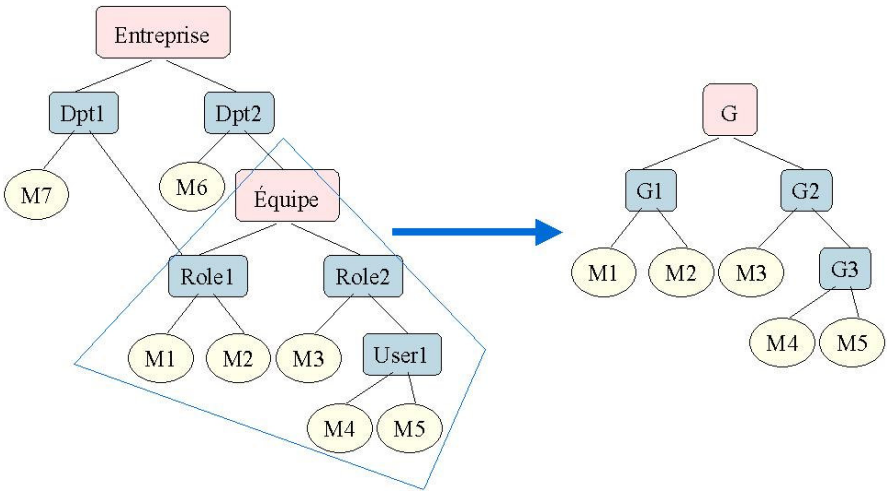

Figure 5.13 Création d'une structure d'entreprise

La racine de la structure est un groupe. Dans le cas d'un déploiement unitaire (une seule machine cible), la racine serait une machine et serait l'unique nœud de la structure. Chaque nœud de type *groupe* peut avoir des fils de type *groupe* (représentant ses sous-groupes) et/ou *machine* (machines du groupe). Les machines sont des feuilles de la structure et n'ont donc aucun fils.

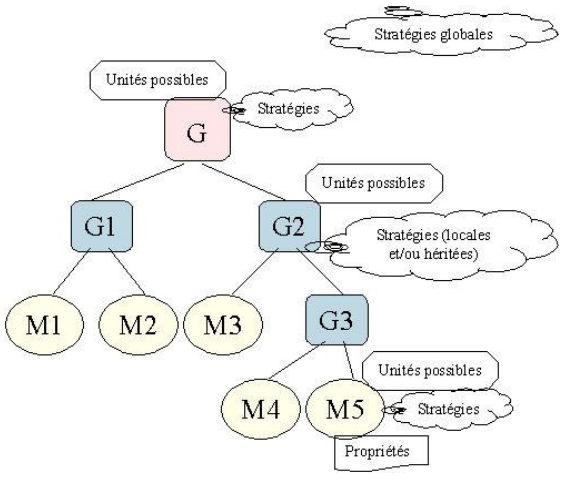

Figure 5.14 Informations liées à la structure d'entreprise

Ensuite, à chacun des nœuds sont associées les informations nécessaires pour le déploiement. Sur la Figure 5.14, seuls quelques nœuds ont de l'information associée afin de ne pas surcharger la figure : dans la réalité, chacun des nœuds peut avoir les différents types d'information.

Ainsi :

- Les machines sont associées à des propriétés.
- A chacun des nœuds peuvent être associées des stratégies. Les stratégies peuvent être :
  - locales, dans le cas où elles sont associées au nœud,
  - héritées, dans le cas où elles ont été transmises par un autre groupe (plus haut dans la structure d'entreprise),
  - globales, dans le cas où elles concernent tous les nœuds de la structure d'entreprise.
- A chaque nœud est aussi associé l'ensemble des unités qui sont possibles pour ce nœud, c'est-à-dire l'ensemble des unités dont les contraintes sont vérifiées, et qui respectent les diverses stratégies de déploiement. Cet ensemble d'unités est représenté par une structure d'application.

5.2. Structure d'application

Une structure d'application représente l'ensemble des unités de déploiement qui sont disponibles (toutes les versions disponibles d'une application). Dans cette structure, une unité de déploiement est représentée par un arbre. Comme le montre la Figure 5.15, la structure d'application est donc une forêt de ces arbres, dont chaque arbre représente l'une des versions de l'application à déployer.

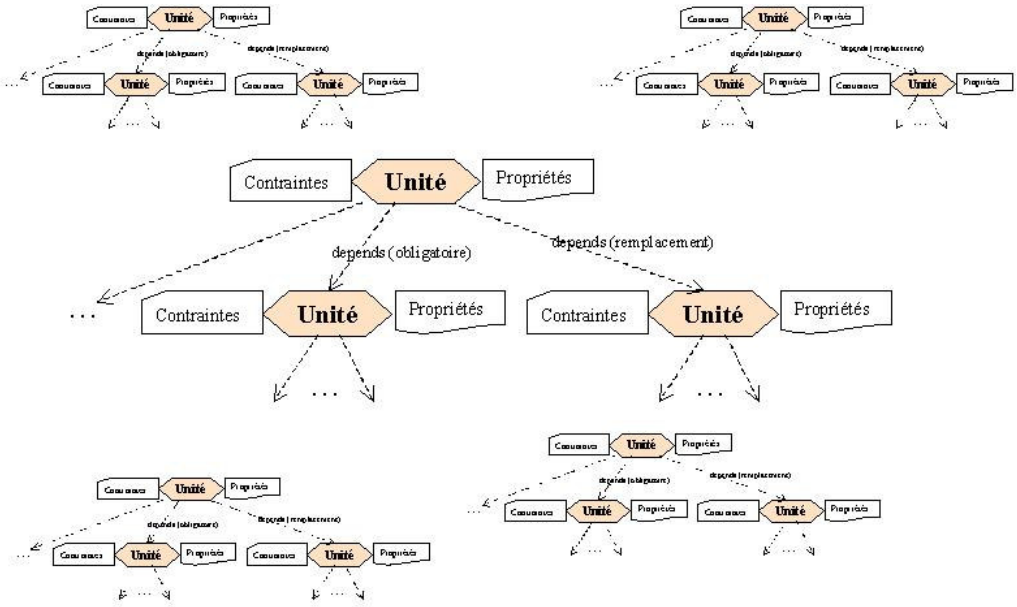

Figure 5.15 Structure d'application

L'un des arbres représentant une unité de déploiement est agrandi dans la Figure 5.15, ce qui permet de mettre en évidence ses différentes caractéristiques. Une unité de déploiement représente une version donnée d'une application. A chaque unité est associée ses contraintes et ses propriétés.

De plus, chaque unité peut avoir des dépendances de plusieurs types. Seules les dépendances obligatoires et les dépendances de remplacement apparaissent sur la figure. Les différents types de dépendances sont matérialisés par le type de lien entre l'unité de déploiement et ses fils. Ainsi, tous les types de dépendances peuvent être représentés dans la structure. Par la suite, nous nous intéressons plus particulièrement aux dépendances obligatoires et de remplacement. Comme chacune des dépendances d'une unité est elle-même une unité, elle peut aussi avoir des dépendances (et ainsi de suite). On obtient alors un arbre qui représente l'unité et tout le chemin des dépendances. Une unité qui n'a pas de fils est une unité qui n'a pas de dépendances.

5.3. Structure de plan

Nous avons vu dans le chapitre précédent que le plan de déploiement permet d'ordonnancer les différents déploiements unitaires. Le plan permet aussi d'exprimer les différents degrés de liberté qui sont encore accordés pour l'exécution du plan (ordre de déploiement partiel, choix d'unités de déploiement, …). Nous proposons ici une structure qui permet de représenter un tel plan.

```
Arité : n, n ≥ 2
Profondeur : m, m ≥ 1
```

La structure de plan est un arbre dont l'arité ($n, n \geq 2$) dépend du nombre de plans unitaires qui sont inclus dans le plan de déploiement global. Sa profondeur dépend ($m, m \geq 1$) du nombre et du type de stratégies d'ordonnancement qui sont appliquées.

```
Arbre -> Pi
Arbre -> /Opérateur, ListeSousArbre\
Opérateur -> parallel | ordered | and | xor | …
ListeSousArbre -> Arbre Arbre | Arbre, ListeSousArbre
```

L'arbre (*Arbre*) représentant la structure de plan peut être définit récursivement. Il est formé d'au moins un plan de déploiement unitaire P_i (cas de base). Les nœuds sont des opérateurs, ayant au moins deux fils (*ListeSousArbre*). Les fils sont soit des sous-arbres (récursivité), soit des feuilles P_i.

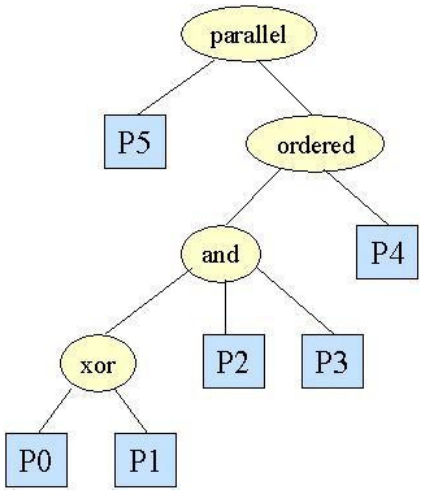

Figure 5.16 Structure de plan

La Figure 5.16 donne un exemple de structure de plan. Une structure de plan est un arbre ayant différents types de nœuds, chacun ayant une sémantique différente. Les feuilles de l'arbre (*Pi*) sont des plans de déploiement unitaires, c'est-à-dire des couples <*unité de déploiement, machine cible*>. Les différents nœuds expriment les sémantiques suivantes :

- ***Parallel*** : l'exécution des plans représentés par les fils doit se faire en parallèle, c'est-à-dire que les différentes exécutions doivent débuter en même temps, et aucune étape ne peut débuter avant la fin des déploiements en parallèle. Le déploiement est alors réalisé de manière synchronisé et peut, par exemple, être utilisé pour le déploiement d'applications distribuées. Dans l'exemple de la Figure 5.16, l'exécution de *P5* est faite en parallèle avec l'exécution du plan représenté par le sous-arbre de gauche.
- ***Ordered*** : l'exécution des plans fils est ordonnée : elle est réalisée depuis le fils gauche, jusqu'au fils droit. Dans l'exemple de la Figure 5.16, pour le sous-arbre dont la racine est le nœud *ordered*, le plan représenté par le sous arbre de gauche s'exécute avant *P4*.
- ***And*** : l'ordre d'exécution des plans fils n'est pas imposé à ce stade. Ce degré de liberté peut éventuellement être utilisé pour satisfaire d'autres stratégies de déploiement (par exemple, selon l'accessibilité des sites cibles au moment de l'exécution), intervenant lors de l'exécution du plan. Dans l'exemple de la Figure 5.16, pour le sous-arbre dont la racine est le nœud *and*, les plans *P2*, *P3* et celui représenté par le sous-arbre de gauche peuvent s'exécuter dans n'importe quel ordre.
- ***Xor*** : un seul des plans fils est exécuté : le choix du plan à exécuter peut dépendre de stratégies à l'exécution du plan. Dans l'exemple de la Figure 5.16, pour le sous-arbre dont la racine est le nœud *xor*, un seul des plans *P0* ou *P1* est exécuté.

De nouveaux types de nœuds peuvent être envisagés pour définir de nouveaux degrés de liberté. Ainsi, par exemple, un nœud pourrait être défini pour exprimer qu'il faut exécuter n plans de déploiement fils parmi un ensemble de p ($p > n$).

De même, d'autres types de nœuds pourraient être ajoutés pour affiner la définition du nœud *parallel* : il peut s'agir :

- d'un déploiement qui débute en parallèle : si des étapes suivent le déploiement *parallel*, elles peuvent débuter dès que l'un des plans à exécuter en parallèle est terminé,
- d'un déploiement qui doit se terminer en même temps : si d'autres étapes suivent, elles doivent attendre la fin de tous les plans à exécuter en parallèle pour débuter leur exécution.

6. MISE EN ŒUVRE DES CONCEPTS COMPLEMENTAIRES AU DEPLOIEMENT

Nous avons vu dans le chapitre précédent que des concepts complémentaires au déploiement étaient utilisés. Les propriétés sont simplement un couple <*nom, valeur*> et n'exigent pas de mise en œuvre particulière ou complexe. Cependant, des dépendances entre unités et des contraintes imposées par les unités doivent aussi être traitées. La mise en œuvre de ces deux concepts exige plus de précisions.

6.1. Dépendances

Les dépendances d'une unité sont conservées dans la structure d'application. Les différents types de dépendances sont représentés comme des attributs de l'unité. Les points suivants énoncent les cas particuliers des différents types de dépendances (cf. paragraphe 3 du Chapitre 4) :

- **Dépendances obligatoires** : Les dépendances obligatoires sont représentées par une liste de liens vers d'autres unités.
- **Dépendances de conflit** : Les dépendances de conflit sont aussi représentées par une liste d'unités. Cette information peut être traitée au moment de la construction du plan de déploiement (avec l'état courant du site cible). Cependant, elle doit aussi être transmise au cours des étapes de préparation, pour être vérifiée à nouveau au moment de l'exécution du plan de déploiement. En effet, si l'état du site a évolué entre la construction et l'exécution du plan (déploiement de l'une des dépendances de conflit, par exemple), une telle dépendance peut provoquer l'échec du déploiement.
- **Dépendances sélectives** : Les dépendances sélectives sont représentées par une liste de listes d'unités. Un élément de chaque liste doit être choisi (et déployé) pour résoudre ces dépendances. Ce choix peut se faire au moment de la sélection des unités à déployer (selon des caractéristiques particulières, par exemple) ou au moment de l'exécution du plan de déploiement (selon les ressources disponibles, par exemple). Ce choix est effectué en accord avec les stratégies de déploiement appliquées par l'entreprise.
- **Dépendances de remplacement** : Les dépendances de remplacement sont représentées par une liste d'unités. En cas de conflit sur une machine cible, ou pour toute autre raison, l'unité de déploiement peut être remplacée par l'une des unités de la liste.

- **Dépendances facultatives** : Les dépendances facultatives sont représentées par une liste d'unités. Ces unités seront déployées ou non, selon ce qu'indiquent les stratégies de déploiement à appliquer. De plus, ces unités ne seront pas déployées s'il y a un problème particulier sur une machine cible, comme par exemple : un conflit avec une unité déjà déployée sur la machine cible, un manque d'espace disque, une stratégie imposant une configuration minimum, etc.

6.2. Contraintes

Les contraintes permettent de vérifier les caractéristiques que doit avoir une machine pour pouvoir recevoir une unité. Elle peuvent être considérés comme des pré-conditions au déploiement. La vérification de certaines d'entre elles est suivie d'une action (décrémenter la propriété indiquant l'espace disque disponible, par exemple).

6.2.1. Langage de contraintes

Pour plus de flexibilité et pour pouvoir définir divers types de contrainte, nous proposons un langage de définition de contraintes. Pour cela, nous décrivons la grammaire BNF correspondante, puis nous donnons quelques exemples permettant de l'illustrer.

Tout d'abord, une contrainte est définie selon un triplet comportant un critère (*expressionBooléenne*), une activité de déploiement (*activitéDeDéploiement*) et une post-action (*action*).

```
Sémantique : ON activitéDeDéploiement
             IF expressionBooléenne THEN action ELSE Abort
```

Le critère (*expressionBooléenne*) est une expression booléenne permettant d'évaluer les caractéristiques et propriétés d'un site client en fonction des besoins de l'unité de déploiement. Cette expression devra être vraie pour que l'unité de déploiement puisse être déployée et fonctionner correctement sur le site cible.

```
< contrainte > ::=
       < activitéDeDéploiement, expressionBooléenne {, action}>

< activitéDeDéploiement > ::= INSTALL | UPDATE | RECONFIGURE |
                  UNINSTALL |ACTIVATE | DESACTIVATE
```

L'activité (*activitéDeDéploiement*) est le nom de l'une des activités du cycle de vie du déploiement. La contrainte n'est appliquée que lors de la réalisation de cette activité. Enfin, la post-action (*action*) est réalisée si le critère (*expressionBooléenne*) est rempli et que l'unité de déploiement est déployée. Si aucune post-action n'est spécifiée, il s'agit simplement de vérifier une condition (cas des contraintes exprimant une sélection, définies précédemment, par exemple).

L'expression booléenne est une expression faisant simplement intervenir des noms de propriétés, des valeurs possibles, des opérateurs logiques et des opérateurs ensemblistes classiques. Dans une prochaine version, cette expression booléenne pourra être remplacée par une expression OCL [Omg03-b]. Ceci permettra d'accéder à des éléments du méta-modèle de déploiement autres que les propriétés des machines.

Lors d'un déploiement, les critères de toutes les contraintes de l'unité de déploiement sont tout d'abord évalués. S'ils sont tous vérifiés, alors les post-actions sont réalisées. Si un seul des critères n'est pas vérifié, cela signifie que l'unité de déploiement ne peut pas fonctionner correctement sur le site cible et le déploiement ne sera pas réalisé.

```
< action > ::= nomFonction ( {listeParamètres} )

< nomFonction > ::= identificateur

< listeParamètres > ::= valeur {, listeParamètres}

< valeur > ::= ChaîneDeCaractères | Réel | 'NomDePropriété'
```

Une post-action (*action*), à réaliser si le critère est vérifié, est exprimée selon le nom d'une fonction. Elle est suivie de la liste de ses paramètres d'entrée si cela est nécessaire. Le code de la fonction est écrit indépendamment et s'exécutera si le critère de la contrainte est vérifié.

Par exemple, une *action* peut désigner une fonction qui permettra de modifier la valeur d'une propriété du site cible. Le paragraphe suivant donne des précisions sur l'utilisation de ce langage.

6.2.2. Quelques exemples

Ce paragraphe a pour objectif de présenter quelques exemples de ce qui peut être fait avec le langage de contraintes que nous proposons.

L'Exemple 1 exprime une contrainte similaire à une contrainte exprimant une quantité dans le modèle de l'OMG (cf. paragraphe 5.2 du chapitre 4). Il s'agit d'indiquer qu'une sortie vidéo est nécessaire. Dans le cas où l'unité concernée est déployée, la propriété de la machine correspondante sera décrémentée de 1.

Exemple 1 :

```
CQ1 = < INSTALL, 'SortiesVideoDisponibles' > 0,
 soustraire('SortiesVideoDisponibles', 1) >
```

L'Exemple 2 exprime que l'unité exige une capacité mémoire supérieure ou égale à *256*.

Exemple 2 :

```
< INSTALL, 'Mémoire' >= 256 >
```

L'Exemple 3 exprime que le système d'exploitation doit être *WindowsXP* ou *Windows2000*.

Exemple 3 :

```
< INSTALL, 'SystèmeDExploitation' ∈ {'WindowsXP',
 'Windows2000'}>
```

Enfin, il est aussi possible de décrire des contraintes plus complexes. L'Exemple 4 exprime que la capacité mémoire d'une machine cible potentielle doit être comprise entre *256* et *512*.

Exemple 4 :

```
< INSTALL, 'Mémoire' >= 256 OR 'Mémoire' >= 512 >
```

7. EXEMPLE DE DEPLOIEMENT SANS STRATEGIES

Dans cette partie, il s'agit de présenter un exemple de mise en œuvre des structures que nous avons définies. Dans cet exemple, nous ne prenons pas encore en compte les stratégies de déploiement, puisqu'elles sont l'objet du chapitre suivant. A ce stade, nous sommes donc simplement capables de déterminer quelles unités sont compatibles avec les configurations de chaque machine. Pour réaliser cette étape (faisant partie de la sélection), il faut utiliser les propriétés et les contraintes.

L'exemple est présenté par complexité croissante. Ainsi, la première partie décrit les structures et ne prend pas en compte les dépendances. Dans un deuxième temps, le traitement des dépendances est ajouté. Enfin, le cumul des contraintes est pris en compte pour terminer cet exemple.

7.1. Exemple de base

L'exemple présenté dans le chapitre précédent mettait en jeu notre laboratoire de recherche. Le modèle d'entreprise défini peut se schématiser par le graphe de gauche de la Figure 5.17. Notre exemple considère maintenant le déploiement d'une application *U* sur les machines du groupe *Adèle*. La structure d'entreprise qui nous intéresse (graphe de droite de la Figure 5.17) peut être déduite du modèle d'entreprise. De plus, au niveau de chaque machine, sont associées les propriétés qui les caractérisent. Sur la figure, celles-ci sont mentionnées sous une forme compréhensible intuitivement, plutôt que formellement.

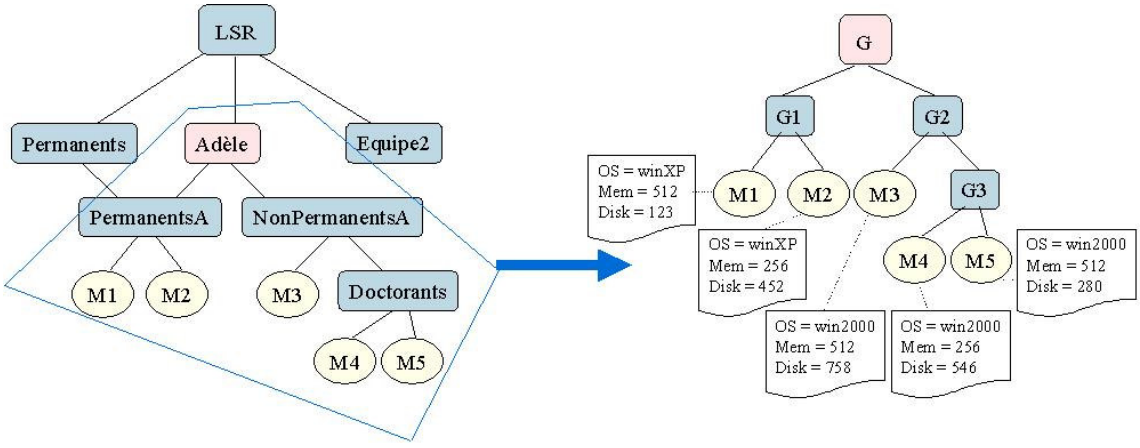

Figure 5.17 Structure d'entreprise pour un déploiement sur le groupe *Adèle*

Nous nous intéressons au déploiement de l'application *U*. Celle-ci est disponible en plusieurs versions. Chacune de ces versions constitue une unité de déploiement, avec ses propres caractéristiques (propriétés et contraintes). L'ensemble de ces unités constitue alors la structure d'application (Figure 5.18) qui concerne le déploiement auquel nous nous intéressons.

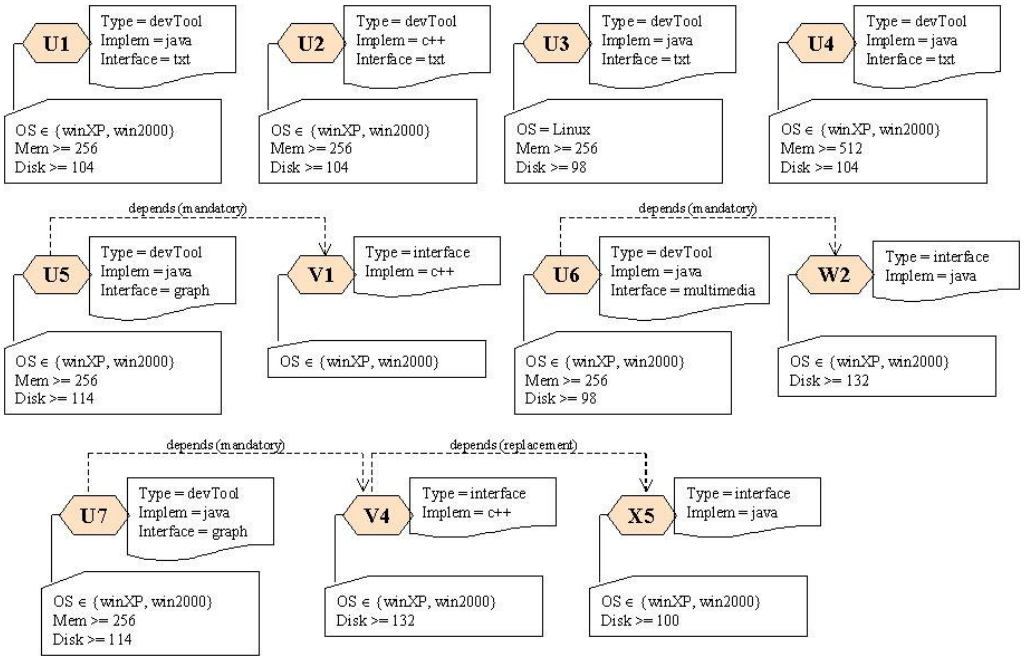

Figure 5.18 Structure d'application des unités de déploiement disponibles

Plusieurs unités sans dépendances sont disponibles : les unités *U1*, *U2*, *U3* et *U4*. Les unités *U5*, *U6* et *U7* sont des unités avec dépendances. Pour *U5* et *U6*, il s'agit de dépendances obligatoires. Quant à *U7*, elle a une dépendance obligatoire vers *V4*, qui a elle-même une dépendance de remplacement vers *X5*. Ceci signifie que *U7* sera déployée soit avec *V4*, soit avec *X5*.

Pour déterminer quelles unités sont « déployables » sur quelles machines, il suffit de vérifier les contraintes imposées par les unités. Pour cela, ce sont les propriétés des machines qui sont utilisées.

Ainsi, par exemple, l'unité *U3* n'est possible pour aucune des machines, puisque aucune n'utilise le système d'exploitation *Linux*. De la même manière, l'unité U4, exigeant *512* de mémoire, ne peut pas être déployée sur les machines n'ayant que *256* (c'est-à-dire *M2* et *M4*).

Machines	Unités de déploiement possibles
M1	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}
M2	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M3	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}
M4	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M5	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}

Figure 5.19 Unités possibles pour chaque machine cible (sans prendre en compte les dépendances)

La Figure 5.19 rassemble les unités possibles pour chaque machine cible à l'issue de cette phase.

7.2. Prise en compte des dépendances

En prenant maintenant en compte aussi les dépendances, il faut s'apercevoir que *W2* et *V4* demandent *132* d'espace disque. Or *M1* ne dispose que de *123*. *W2* ne peut donc pas être déployée sur *M1*, et ceci supprime *U6* de la liste des possibilités pour la machine. L'unité *V4* ne peut pas non plus être déployée sur la machine *M1* et est supprimée de la liste des possibilités. Mais cette unité a une dépendance de remplacement vers *X5*, dont les contraintes sont vérifiées par la machine. *U7* est donc conservée dans la liste des possibilités, mais avec *X5* pour seule possibilité de résolution de dépendance.

En réalisant la vérification des contraintes de chaque unité de déploiement, pour chaque machine cible, le tableau de la Figure 5.20 est obtenu. Ce tableau rassemble toutes les unités qui sont « déployables » sur chaque machine.

Machines	Unités de déploiement possibles
M1	E3 = {U1, U2, U4, U5(V1), U7(X5)}
M2	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M3	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}
M4	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M5	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}

Figure 5.20 Unités de déploiement possibles pour chaque machine cible (contraintes traitées indépendamment)

7.3. Cumul des contraintes

Dans les parties précédentes, les contraintes ont été présentées et traitées individuellement. Cependant, nous avons vu dans le chapitre précédent que des contraintes différentes peuvent faire intervenir les mêmes propriétés des machines cibles. Dans le cas où les valeurs de ces propriétés sont modifiées par la post-action d'une contrainte, les contraintes ultérieures doivent prendre en compte la nouvelle valeur.

Dans notre exemple, pour une unité donnée, chaque contrainte fait intervenir des propriétés différentes. Cependant, certaines unités ont des dépendances. Dans ce cas, les mêmes propriétés peuvent être utilisées plusieurs fois : c'est le cas de la propriété exprimant l'espace disque disponible. Ainsi, pour le déploiement de l'unité *U6* et de sa dépendance *W2*, une machine cible potentielle doit avoir un espace disque d'au moins *98 + 132 = 230*. De la même manière, pour le déploiement d'*U7* et de sa dépendance *V4*, il faut *114 + 132 = 246* ; et pour le déploiement de *U7* et de *X5*, *114 + 100 = 214* sont nécessaires.

Le cumul des contraintes impose alors de supprimer *U7* de la liste des possibilités pour la machine *M1*. Le tableau de la Figure 5.21 contient l'ensemble des unités possibles pour chaque machine.

Machines	Unités de déploiement possibles
M1	E4 = {U1, U2, U4, U5(V1)}
M2	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M3	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}
M4	E2 = {U1, U2, U5(V1), U6(W2), U7({V4, X5})}
M5	E1 = {U1, U2, U4, U5(V1), U6(W2), U7({V4, X5})}

Figure 5.21 Unités de déploiement possibles pour chaque machine cible (cumul de contraintes)

La Figure 5.21 résume simplement l'information contenue dans la structure de plan. La Figure 5.22 illustre cette structure.

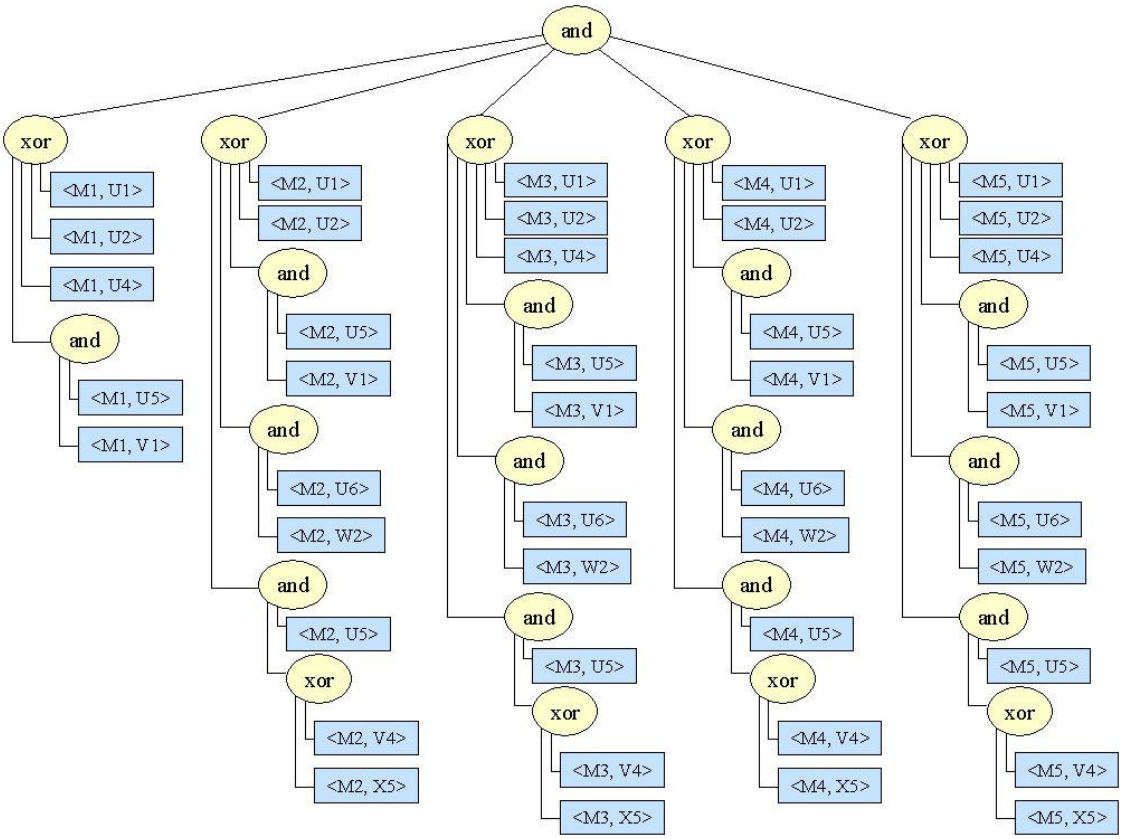

Figure 5.22 Structure de plan pour le déploiement de l'application *U* sur les machines du groupe *Adèle*

Avant l'application des stratégies, il n'est pas possible d'établir l'ordre de déploiement entre les différents plans unitaires : la racine de l'arbre est donc simplement un nœud de type *and*. Au niveau inférieur, on retrouve ensuite un nœud *xor* qui indique quelles unités sont possibles pour chaque machine. Lorsqu'une unité a des dépendances obligatoires, un nœud *and* indique le déploiement de l'unité et de sa dépendance. Les dépendances sélectives sont, quant à elles, exprimées par un nœud *xor*.

8. SYNTHESE

Différentes technologies peuvent être utilisées pour réaliser le déploiement. La technologie des procédés permet de structurer et d'automatiser les différentes étapes du déploiement. Elle permet aussi de gérer les cas où des erreurs se produisent. La technologie des fédérations, quant à elle, permet de lier les concepts des différents domaines utilisés dans le déploiement. Elle apporte aussi de la flexibilité aux entreprises, puisque cette technologie peut être utilisée avec des modèles spécifiques différents.

Notre environnement de déploiement ORYA est basé sur l'utilisation de ces technologies. Les concepts du déploiement ont été répartis en différents domaines (éventuellement déjà existants). La composition de ces différents domaines forme le domaine de déploiement définissant tous les concepts identifiés précédemment. Le plan de déploiement, lui aussi, est géré par un procédé.

Des structures ont été définies pour traiter l'information de déploiement et créer le plan de déploiement global permettant de déployer un ensemble d'unités sur un ensemble de machines :

- La **structure d'entreprise** permet de décrire la structure hiérarchique (groupes, sous-groupes et machines), cible du déploiement.
- La **structure d'application** décrit l'ensemble des unités de déploiement possibles pour une machine ou un groupe.
- La **structure de plan** permet d'ordonnancer les différents plans de déploiement unitaires pour former le plan de déploiement global.

Enfin, un langage de contraintes a été conçu pour permettre à l'utilisateur de définir lui-même les types de contraintes qu'il souhaite exprimer.

CHAPITRE 6 :
UN ENVIRONNEMENT DE DEPLOIEMENT ORIENTE STRATEGIES

Préambule :

Nous avons vu jusqu'à présent comment automatiser le déploiement à grande échelle. La problématique de ce travail est destinée aux entreprises. Celles-ci peuvent aussi souhaiter personnaliser leurs déploiement en fonction de différentes choses : un ordonnancement particulier, une préférence de types d'application, la bande-passante disponible, la sécurité… Il s'agit alors de stratégies de déploiement. La définition et la mise en œuvre des stratégies sont traitées dans ce chapitre.

---

1. INTRODUCTION : NOTRE APPROCHE

Le déploiement à grande échelle est une phase très complexe du cycle de vie du logiciel. Cette étape ne peut raisonnablement pas être réalisée à la main. C'est pourquoi un support automatique est indispensable pour cette phase.

De plus, les entreprises souhaitent pouvoir appliquer des stratégies (ou politiques) de déploiement. Des stratégies permettent d'assurer un certain niveau de qualité à l'opération, que ce soit en termes de sécurité, d'homogénéité, de conformité aux standards, etc. Certaines approches incluent directement « en dur » les stratégies dans les outils de déploiement. Ceci implique que l'utilisateur ne peut pas définir de nouvelles stratégies mieux adaptées à ses besoins.

Une stratégie peut aussi être vue comme une contrainte, exprimée sur l'environnement cible (structure d'entreprise), concernant les unités à déployer. Notre objectif est de fournir un moyen d'exprimer des stratégies de déploiement avancées. L'utilisateur peut ainsi définir ses propres stratégies, en fonction de ses besoins.

Nous avons choisi d'associer des stratégies aux groupes et machines d'une entreprise. En effet, le choix des stratégies appartient à l'entreprise (cible de déploiement). Il semble donc logique de lier les stratégies aux entités de l'entreprise (groupes et machines).

Chaque stratégie est appliquée à un ensemble d'unités « déployables ». Ces unités sont représentées par une structure d'application. Le résultat de l'application des stratégies est une structure de plan qui contient tous les déploiements unitaires (couples < unité, machine>) qui doivent être réalisés. Cette structure de plan est construite à partir de la structure d'application (unités possibles), de la structure d'entreprise (cible de déploiement) et des stratégies qui peuvent être appliquées à divers moments de la construction de cette structure.

Nous pouvons considérer une stratégie comme étant constituée d'un triplet <*activitéDeDéploiement, contrainte-Unités, contrainte-Entreprise*>. L'activité de déploiement (*activitéDeDéploiement*) représente l'activité durant laquelle s'applique la stratégie.

Une expression logique sur les propriétés des unités (*contrainteUnités*) permet de déterminer pour quel type d'unités la stratégie doit s'appliquer. Cette expression permet de séparer l'ensemble des unités possibles en deux sous-ensembles : celui des unités qui évaluent la *contrainte-Unités* à *vrai* et celui des unités qui l'évaluent à *faux*. Selon la stratégie, des traitements peuvent être appliqués sur l'un ou l'autre de ces ensembles, éventuellement les deux. Le résultat de ce traitement est le résultat de l'application de la stratégie.

Enfin, le champ *contrainte-Entreprise* représente le corps de la stratégie, en exprimant une contrainte de déploiement (ordre de déploiement, choix d'une même version pour un groupe de machine, etc.) imposée par l'entreprise. Il représente la stratégie elle-même et permet de décider quel(s) traitement(s) appliquer aux sous-ensembles, pour construire l'ensemble d'unités, résultat de l'application de la stratégie. D'une certaine manière, ce champ définit la sémantique de base de la stratégie.

Les stratégies peuvent exprimer diverses préférences de l'utilisateur. Ainsi, plusieurs exemples de stratégies peuvent être donnés :

- déployer la même unité (version d'une application) sur un ensemble de machines,
- permettre le remplacement d'une unité par une autre correspondant à une version plus récente,
- privilégier le déploiement d'unités ayant certaines caractéristiques (implémentation en java, par exemple),
- déployer les dépendances de l'unité avant l'unité elle-même,
- déployer l'unité sur un groupe de machines avant de la déployer sur un autre groupe (ou sur une machine avant une autre machine),
- effectuer un retour en arrière en cas d'erreur (dues à l'évolution de l'environnement) au cours de l'exécution du plan de déploiement,
- etc.

Nous verrons dans la suite de ce chapitre qu'une stratégie a un type qui définit ses caractéristiques et qu'elles peuvent être regroupées en catégories.

Les stratégies peuvent être utilisées à tout moment du cycle de vie du déploiement. Dans notre travail nous nous sommes intéressés plus particulièrement aux stratégies liées au déploiement initial d'une unité. De plus, pour simplifier le discours, nous nous spécialisons dans la suite de ce document au cas du déploiement d'une application (ensemble d'unités possibles) sur un groupe de p machines. Il s'agit du déploiement de n exemplaires d'une même application (éventuellement plusieurs versions) sur un ensemble de p machines, et non du déploiement d'une seule application distribuée sur p machines. Il est ensuite assez simple de passer au cas plus général du déploiement de n applications sur un groupe de p machines.

Dans ce chapitre, après une brève modélisation, nous indiquons les différentes catégories de stratégies et leurs caractéristiques, avec quelques exemples pour les illustrer. Puis, nous présentons l'algorithme de base qui permet de mettre en œuvre les stratégies. Un langage de stratégies est ensuite défini. Enfin, nous terminons par une discussion sur les interactions possibles avec les stratégies.

2. MODELISATION DE STRATEGIE

Dans le méta-modèle de déploiement (Figure 4.2), nous avons vu qu'il existe une relation entre une stratégie et l'entité de l'entreprise (machine ou groupe) qui l'applique. Au moment du déploiement, ces stratégies sont utilisées pour construire un plan de déploiement, qui permet de déployer des unités sur un groupe de machines. Des associations sont alors créées, à l'exécution, entre :

- les entités de l'entreprise (groupe et/ou machines),
- l'unité à déployer,
- les stratégies qui sont utilisées pour construire et exécuter le plan de déploiement global, et,
- le plan de déploiement qui est construit puis exécuté.

Ces associations permettent d'exprimer avec quelles stratégies un plan de déploiement global a été construit, et quelles unités peuvent être déployées sur quelles machines en l'utilisant.

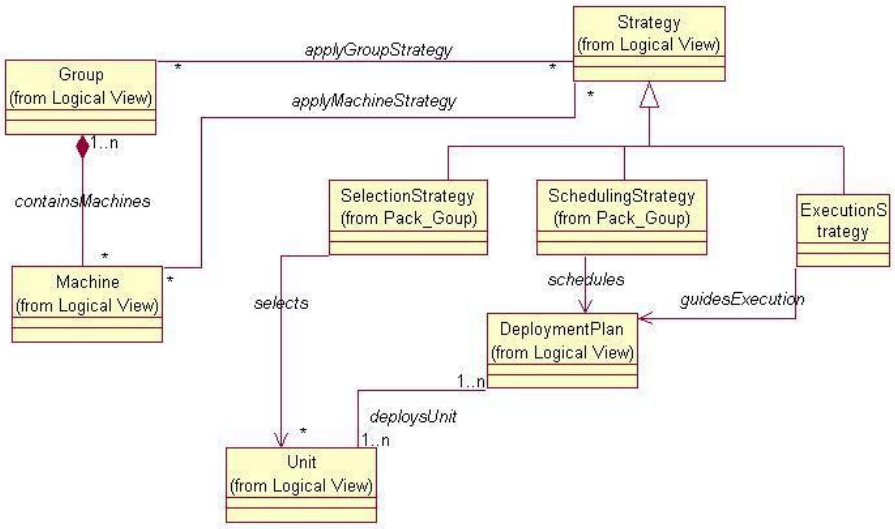

Figure 6.1 Modèle de stratégie et ses relations

La Figure 6.1 présente le modèle de stratégie et ses relations. La partie de gauche (concepts de *Group* et *Machine*) a déjà été présentée dans le chapitre 4 (Figure 4.3). Les stratégies sont associées à l'entité de l'entreprise à qui les appliquer, à savoir un groupe (*applyGroupStrategy*) ou une machine (*applyMachineStrategy*).

Le reste de la Figure 6.1 illustre les relations qui sont créées lorsqu'un déploiement est demandé. Les stratégies permettent alors de construire un plan de déploiement global. Elles peuvent être de différentes sortes (*SelectionStrategy*, *SchedulingStrategy*, *ExecutionStrategy*, …) et sont utilisées pour sélectionner les unités à déployer (*selects*), pour ordonnancer le plan de déploiement (*schedules*) ou pour guider son exécution (*guidesExecution*). L'exécution de ce plan permet de déployer un ensemble d'unités (*deploysUnit*) sur un ensemble de machines.

La gestion de ce genre de liens permet notamment de conserver un historique de tout ce qui s'est passé, en terme de déploiement, sur une entité de l'entreprise. Ces liens peuvent aussi permettre, par exemple, de pouvoir configurer une nouvelle machine en fonction des stratégies déjà appliquées dans le groupe où elle est insérée.

3. CATEGORIES, TYPES ET CARACTERISTIQUES DES STRATEGIES

Les stratégies de déploiement sont utilisées pour construire un plan de déploiement global. Elles peuvent intervenir à différentes phases de sa construction. Nous pouvons alors définir différentes catégories de stratégies. Ensuite, chaque stratégie peut être associée à un ensemble de caractéristiques qui lui sont propres [CLM05].

3.1. Catégories et types de stratégies

Différentes catégories de stratégies interviennent à divers moments de la construction du plan de déploiement. Ainsi, l'utilisation des stratégies permet de :

- **Sélectionner les unités à déployer** (Où et quoi déployer ?) : Il s'agit de déterminer quelle(s) unité(s) déployer sur quelle(s) machine(s).
- **Définir le plan de déploiement** (Quand et comment déployer ?) : Il s'agit d'ordonner les différentes étapes (déploiements unitaires) à réaliser pour effectuer le déploiement complet. Ces actions incluent le déploiement des dépendances de l'unité ou non. De plus, l'ordonnancement obtenu peut être total ou partiel.
- **Personnaliser l'exécution du plan de déploiement :** Il s'agit principalement d'ajouter le traitement des erreurs. Il s'agit par exemple des erreurs qui peuvent apparaître lors de l'exécution du plan de déploiement, notamment à cause de l'évolution des sites depuis la construction du plan de déploiement. Il peut aussi s'agir de fixer les différents degrés de libertés qui n'ont pas été fixés lors des étapes précédentes (sélection incomplète des unités ou cibles de déploiement).

Une fois l'ensemble des stratégies appliquées, un plan personnalisé de déploiement global est créé. Ce plan permet de déployer une unité sur un groupe de machines. Chaque machine concernée reçoit sa propre version de l'application (éventuellement la même).

A partir de ces différentes phases de la construction d'un plan de déploiement global, nous pouvons établir une classification en trois catégories de stratégies (cf. le modèle de stratégie ci-dessus) :

- les **stratégies de sélection** (*SelectionStrategy*),
- les **stratégies d'ordonnancement** (*SchedulingStrategy*), et,
- les **stratégies d'exécution** (*ExecutionStrategy*).

Chacune de ces catégories de stratégies est liée à la phase de la construction/exécution d'un plan de déploiement global, durant laquelle elles sont utilisées.

Chaque catégorie peut contenir plusieurs **types** de stratégies. Ainsi, par exemple, pour les stratégies de sélection, les types de stratégies suivants peuvent être énoncés :

- définir la compatibilité qui doit exister entre les différentes machines d'un groupe,
- caractériser les droits à déployer ou non un certain type d'unité.

D'autres types de stratégies seront définis et illustrés plus en détails dans la partie suivante.

Plusieurs caractéristiques permettent ensuite de déterminer plus précisément la sémantique de chaque type de stratégie.

3.2. Caractéristiques des stratégies

Chaque stratégie est définie par son comportement basique et des caractéristiques. Les stratégies d'un même type ont leurs caractéristiques en commun. Ces caractéristiques doivent être spécifiées pour définir la sémantique de la stratégie. Nous identifions les caractéristiques suivantes :

- La **position** détermine à quel(s) type(s) de nœud peut être associée la stratégie. En effet, une stratégie peut être attachée à une machine ou à un groupe de machines. Certains types de stratégies peuvent être associés aux deux types de nœud, d'autres ne peuvent être associés qu'à un seul type de nœud.
- La **portée** détermine à quelle(s) cible(s) doit s'appliquer la stratégie. Ainsi, une stratégie attachée à une machine ne concerne que la machine elle-même. Une stratégie associée à un groupe, quant à elle, concerne toutes les machines du groupe ou certaines machines du groupe. Dans les cas simples, la portée peut être déduite de la position. C'est ce que nous ferons comme hypothèse lors de la présentation de la grammaire du langage de stratégies (cf. paragraphe 6 de ce chapitre)
- La **visibilité** détermine si la stratégie doit ou non cacher les stratégies de même type des niveaux inférieurs (sous-groupes et machines) de la structure d'entreprise. En effet, une stratégie associée à un groupe peut ou non cacher (ou être cachée) par une stratégie de même type liée à un des sous-nœuds.
- La **propagation** indique si la stratégie doit ou non être propagée aux sous-nœuds de la structure d'entreprise. En effet, avant de pouvoir être appliqués, certains types de stratégies peuvent par exemple exiger une collecte d'information en provenance des niveaux inférieurs. Dans ce cas, la stratégie peut être propagée ou non aux sous-nœuds. Cette propagation aux sous-nœuds peut dépendre du type des sous-nœuds : par exemple, la stratégie peut n'être propagée qu'aux sous-groupes (et pas aux machines).
- La **précédence** détermine quelles stratégies doivent être appliquées en priorité. En effet, plusieurs stratégies de même type peuvent être appliquées à un même nœud. Il faut alors définir si ce sont les stratégies locales ou les stratégies héritées (propagées) voire globales qui sont prioritaires.

Lors de la définition d'un nouveau type de stratégie (quelque soit sa catégorie), ses caractéristiques doivent alors être spécifiées. Elles permettent de déterminer la manière d'appliquer la stratégie. A ces informations s'ajoute la sémantique de base de la stratégie. Celle-ci permet alors de connaître quel(s) traitement(s) appliquer aux sous-ensembles d'unités possibles. Ce traitement permet de déterminer quelles unités possibles doivent être conservées.

4. DES EXEMPLES DE TYPES DE STRATEGIES

Trois catégories de stratégies ont été identifiées dans la section précédente. Dans chaque catégorie, nous avons défini plusieurs types de stratégies. Nous présentons ici quelques exemples de types de stratégies, que nous avons identifiés, pour chacune de ces catégories. Pour chacun d'eux, nous définissons leurs caractéristiques. Nous pouvons noter que ces types

de stratégies ne sont pas exhaustifs et peuvent être étendus (en proposant de nouveaux choix, par exemple).

Pour simplifier la présentation de ces exemples, nous définissons deux sous ensembles de l'ensemble des unités possibles :

- L'ensemble *ensemble-Vrai* est le sous-ensemble composé des unités qui vérifient la contrainte sur les unités (*contrainte-Unités*) de la stratégie.
- L'ensemble *ensemble-Faux* est le sous-ensemble composé des unités qui ne vérifient pas la contrainte sur les unités de la stratégie.

L'union de ces deux ensembles constitue l'ensemble des unités possibles, avant l'application de la stratégie. En fonction de la stratégie, des traitements sont appliqués sur l'un ou l'autre de ces sous-ensembles (ou sur les deux), pour déterminer l'ensemble des unités à conserver. Les unités de cet ensemble résultat constituent les unités contenues dans la structure d'application (cf. le paragraphe 5.2 du chapitre 5).

4.1. Stratégies de sélection

Nous présentons trois types de stratégies de sélection. Elles permettent de déterminer quelle(s) unité(s) doivent être déployées sur chaque machine ou groupe de machines.

4.1.1. Stratégies VERSION-DROITS

Les stratégies de type *VERSION-DROITS* permettent d'imposer ou d'interdire le déploiement d'un certain type d'unités. Par exemple, l'entreprise peut imposer le déploiement d'applications implémentées en java, ou interdire le déploiement de jeux, … Ces stratégies peuvent être associées à une machine (dans ce cas, elle ne concernent qu'une seule cible) ou à un groupe de machines (elles concernent toutes les machines cibles du groupe).

Ce type de stratégie ne nécessite pas d'information supplémentaire (provenant des nœuds fils), elles ne sont donc pas propagées et sont traitées directement. Nous avons défini les choix (champ *contrainte-Entreprise* du triplet représentant une stratégie) suivants pour ce type de stratégie :

- **ALWAYS** : Cela signifie que seulement les unités de l'*ensemble-Vrai* peuvent être déployées. La structure d'application, résultat de l'application de la stratégie, est composée des unités de l'*ensemble-Vrai*.
- **NEVER** : Cela signifie que les unités de l'*ensemble-Vrai* ne peuvent pas être déployées. La structure d'application, résultat de l'application de la stratégie, est composée des unités de l'*ensemble-Faux*.
- **IF-AVAILABLE** : Cela indique une préférence pour les unités contenues dans l'*ensemble-Vrai*, mais pas une obligation. Dans ce cas, si l'*ensemble-Vrai* n'est pas vide, c'est lui qui compose la structure d'application, résultat de l'application de la stratégie. S'il est vide, la structure d'application est composée des unités de l'*ensemble-Faux*. Si aucune des unités de l'*ensemble-Vrai* n'est finalement compatible avec les caractéristiques d'une (ou plusieurs) machine(s) cible(s), une nouvelle tentative est réalisée avec les unités de l'*ensemble-Faux*.

4.1.2. Stratégies VERSION-INTRAGROUPE

Les stratégies de type *VERSION-INTRAGROUPE* permettent d'exprimer quelle compatibilité doit exister entre les différentes machines d'un groupe. Ce type de stratégie ne peut donc être appliqué qu'aux nœuds de type groupe et concerne la compatibilité d'unité entre toutes les machines du groupe. Il s'agit ici de décider si toutes les machines d'un groupe doivent avoir la même version d'une application (unité de déploiement) ou si elles peuvent avoir des versions différentes, selon leurs caractéristiques spécifiques.

Le traitement de ces stratégies nécessitent de l'information (unités possibles) en provenance des nœuds fils. En effet, pour connaître quelle unité peut être déployée sur toutes les machines du groupe, il faut connaître pour chaque machine quelles unités sont compatibles avec la configuration de cette machine. Ces stratégies sont propagées aux sous-nœuds de type groupe.

Au niveau visibilité, une stratégie de ce type cache les stratégies de même type des sous-groupes. La priorité est donc accordée aux stratégies héritées plutôt qu'aux stratégies locales.

Nous avons défini les choix (champ *contrainte-Entreprise* du triplet représentant une stratégie) suivants pour ce type de stratégie :

- **UNIT** : Cela signifie que la même unité (même version d'une application) doit être déployée sur toutes les machines du groupe. Dans ce cas, c'est une unité de l'*ensemble-Vrai* qui doit être choisie. La stratégie est donc propagée, avec la structure d'application composée des unités de l'*ensemble-Vrai*, aux fils de type groupe. Une fois que toute l'information en provenance des fils est disponible, c'est-à-dire une fois que les sous-ensembles des possibilités pour chaque machine cible sont connus, la stratégie peut être appliquée. Il s'agit alors de prendre l'intersection des ensembles de possibilités de chaque machine.
- **UNIT-IF-POSSIBLE** : Cela signifie que les machines doivent recevoir la même unité de l'*ensemble-Vrai* ou une unité différente de l'*ensemble-Faux*. Dans ce cas, c'est d'abord la structure composée des unités de l'*ensemble-Vrai* qui est propagée aux fils. Si après le traitement de la stratégie, aucune unité n'est possible, alors, la structure d'application composée de l'*ensemble-Faux* est propagée à son tour pour déterminer une unité pour chaque machine.

Dans le cas où aucune stratégie de ce type n'est appliquée à un groupe, cela signifie que des unités différentes peuvent être déployées sur chaque machine cible du groupe.

4.1.3. Stratégies VERSION-UNITE

Les stratégies de type *VERSION-UNITE* permettent d'exprimer quelle compatibilité doit exister entre les unités de même nom (versions d'une même application) sur une machine. Par exemple, l'entreprise peut interdire le remplacement d'une version d'une application par une autre, afin de conserver la compatibilité de versions entre plusieurs machines. Dans le cas où ce type de stratégie est appliqué à un groupe, il concerne toutes les machines du groupe. S'il est appliqué à une seule machine, il ne concerne que cette cible.

Ce type de stratégie est propagé jusqu'aux nœuds de type machine, puisque ce sont les machines qui sont les cibles finales et qui sont concernées par ces informations. Les stratégies

héritées sont prioritaires sur les stratégies locales. De plus, les stratégies de ce type cachent les stratégies de même type des nœuds fils.

Les choix possibles sont basés sur le modèle pour les applications à base de composants [PDC01] que nous avons présenté dans le chapitre 3. Les contraintes d'entreprise suivantes sont possibles pour ce type de stratégie :

- **IS** (*INSERT-SEPARATELY*) : Cela signifie que deux unités de même nom peuvent être déployées sur la machine.
- **NR** (*NEVER-REPLACE*) : Cela signifie que si une unité est déjà déployée sur la machine, alors elle ne peut pas être remplacée par (ni cohabiter avec) une unité de même nom.
- **RA** (*REPLACE-ALWAYS*) : Même si une unité est déjà déployée, elle doit être remplacée par une autre unité de même nom (peu importe son numéro de version).
- **ROIN** (*REPLACE-ONLY-IF-NEWER*) : Si une unité est déjà déployée, elle doit être remplacée par une unité de même nom, seulement si celle-ci a une version supérieure à la première.

Ces stratégies peuvent être utilisées lors de la sélection d'une unité, mais elles peuvent aussi être utilisées lors de l'exécution du plan de déploiement. En effet, si, entre la construction du plan et son exécution, une unité a été déployée et qu'il faut appliquer la stratégie *NR*, alors le déploiement de l'unité ne pourra pas être réalisé .

4.2. Stratégies d'ordonnancement

Nous présentons deux types de stratégies d'ordonnancement. Elles permettent de déterminer dans quelle ordre doivent s'exécuter les différents plans de déploiement unitaires (couple <machine, unité>). Ainsi, après l'application des stratégies d'ordonnancement, le plan de déploiement global est ordonné (partiellement ou totalement).

Une fois les stratégies de sélection appliquées, chaque nœud de la structure d'entreprise est associé à la structure d'application formée des unités possibles pour ce nœud. Il reste ensuite à effectuer un autre parcours de la structure d'entreprise pour créer le plan de déploiement, ordonné selon l'application des stratégies d'ordonnancement.

Les stratégies d'ordonnancement ne concernent que l'ordonnancement du déploiement des unités qui vérifient la *contrainte-Unités* de la stratégie.

4.2.1. Stratégies CHRONOLOGIE-MACHINES

Les stratégies de type *CHRONOLOGIE-MACHINES* permettent d'exprimer dans quel ordre doivent être réalisés les déploiements unitaires de chaque machine. Cet ordre indique des noms de sous-groupes et/ou machines. Ce type de stratégie ne s'applique qu'aux groupes. Il permet par exemple d'exprimer que le déploiement d'un traitement de texte doit d'abord être réalisé sur la machine M1, puis sur la machine M2. Il peut aussi exprimer de débuter par la machine M1, puis de continuer avec les machines du sous-groupe G4.

En ce qui concerne la visibilité, les stratégies de ce type ne cachent pas les stratégies de même type des nœuds-fils (sous-groupes). Cependant, une priorité est accordée au niveau du nœud (groupe) père. En fait, les stratégies de *CHRONOLOGIE-MACHINES* des nœuds fils permettent d'affiner récursivement l'ordre de déploiement.

Il n'est pas nécessaire de propager ce type de stratégie. Cependant des données issues des nœuds fils (sous arbre de la structure d'entreprise) sont nécessaires pour définir intégralement l'ordre de déploiement.

Le résultat de l'application des stratégies de chronologie peut ne définir qu'un ordre partiel de déploiement. Mais, pour un nœud donné, l'ordre concerne tous les fils directs (sous-groupes et machines) du groupe. En effet, soit un ordre total est déterminé (nœud *ordered* ou *parallel* de la structure de plan), soit aucun ordre n'est déterminé (nœud *and* de la structure de plan). C'est ce deuxième cas qui conduit à un ordre partiel de déploiement en cumulant les stratégies de chronologie des différents niveaux. Les mots-clés suivants peuvent intervenir dans la *contrainte-Entreprise* de ce type de stratégie :

- **PARALLEL** : Les déploiements unitaires doivent être réalisés en parallèle, de manière synchronisée. Dans ce cas, les déploiements commencent en même temps, et doivent tous être terminés avant d'effectuer l'étape suivante du plan de déploiement.
- **ORDERED** : Les déploiements unitaires sont réalisés séquentiellement selon un ordre précis.
- **AND** : Les déploiements unitaires sont réalisés séquentiellement sans ordre précis.

Ces mots-clés sont suivis d'une liste de noms de groupes et/ou machines (fils directs du nœud auxquels est appliquée la stratégie). Selon le choix indiqué, le nœud correspondant est créé pour la structure de plan. Ainsi, par exemple, l'utilisation du mot-clé *PARALLEL* conduira à la création d'un nœud *parallel* dans la structure de plan. La Figure 6.2 indique la structure de plan qui est construite à partir de deux stratégies de type *CHRONOLOGIE-MACHINES*, pour le déploiement de l'unité *Z3* sur les machines du groupe *H1*. Le déploiement parallèle sur les machines *M7* et *M8* débute alors après la déploiement sur la machine *M6*.

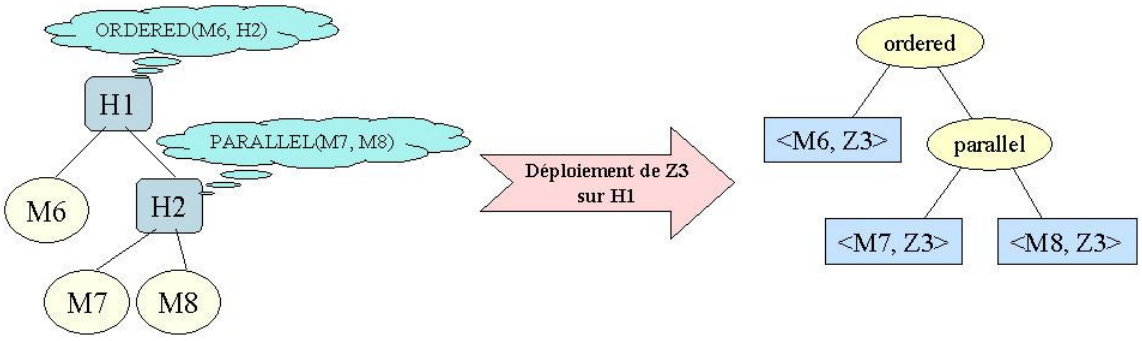

Figure 6.2 Exemple d'ordonnancement

La liste des machines et sous-groupes peut aussi être remplacée par un entier. Ceci indique que le déploiement ne doit pas avoir lieu sur tous les fils du groupe, mais seulement sur une

partie. Ainsi, par exemple *AND(2)* signifie que l'unité de déploiement doit être déployée sur deux des fils du groupe, sans ordre précis. Un groupe pouvant avoir des fils de type groupe ou machine, l'entier indique le nombre de fils concernés :

- S'il n'y a que des fils machines, cet entier correspond au nombre de déploiements unitaires.
- S'il n'y a que des sous-groupes, l'entier correspond au nombre de sous-groupes concernés.
- S'il y a des machines et des sous-groupes, l'entier correspond au nombre de fils (machines ou groupes) concernés.

Cette notation permet de remplacer les choix de type *1IN*, *2IN*, etc., indiquant que seulement quelques déploiements sont nécessaires. Cependant, lors de la création de la structure de plan, ce choix est indiqué par un nœud *xor*. Cette notation permet de mentionner la notion de choix (qui doit être résolu au moment de l'exécution du plan de déploiement), mais fait perdre de l'information. En effet, c'est la même notation qui est utilisée lorsque plusieurs unités sont encore possibles. Lorsque l'exécution du plan arrive sur un tel nœud de choix, il est alors impossible de savoir si le choix doit être orienté par les caractéristiques des machines ou des unités. Cette notation permet cependant de représenter les degrés de libertés du plan de déploiement de manière assez simplifiée, mais devra évoluer. En effet, il est particulièrement important de faire évoluer cette notation lors du passage au déploiement de *n* applications sur *p* machines (ici, nous traitons le cas d'une application sur *p* machines). Dans ce cas, il faudra instaurer une notation permettant d'indiquer le nombre d'unités souhaitées et le nombre de machines cibles exigés.

4.2.2. Stratégies CHRONOLOGIE-UNITES

Les stratégies de type *CHRONOLOGIE-UNITES* permettent d'exprimer dans quel ordre doivent être déployées différentes unités. Comme nous travaillons particulièrement sur le cas du déploiement d'une application sur *p* machines, ce type de stratégie concerne essentiellement l'ordre de déploiement entre l'unité et ses dépendances. Il s'agit alors de déterminer si les dépendances d'une unité de déploiement doivent être déployées avant ou après ou en même temps que celle-ci. Si ce type de stratégie est appliquée à un groupe, tous les déploiements unitaires associés aux machines de ce groupe sont concernés. Si la stratégie est appliquée à une machine, elle ne concerne que cette machine.

En ce qui concerne la visibilité, les stratégies de ce type cachent les stratégies de même type des nœuds-fils (sous-groupes). Ces stratégies sont propagées jusqu'aux nœuds machines (cibles finales de déploiement).

Les mots-clés d'ordonnancement définis pour les stratégies CHRONOLOGIE-MACHINES sont utilisés. Les listes de machines (ou sous-groupes) sont remplacées par les mots clés suivants :

- **MANDATORY-DEPENDENCIES** : Ce mot-clé fait référence au déploiement des dépendances obligatoires d'une unité.

- **SELECTIVE-DEPENDENCIES** : Ce mot-clé fait référence au déploiement des dépendances sélectives d'une unité.
- **OPTIONAL-DEPENDENCIES** : Ce mot-clé fait référence au déploiement des dépendances facultatives d'une unité.
- **UNIT** : Ce mot-clé fait référence au déploiement de l'unité elle-même.

Ainsi, par exemple, une stratégie dont la *contrainte-Entreprise* est *ORDERED(MANDATORY-DEPENDENCIES, UNIT)* indique qu'il faut d'abord déployer les dépendances obligatoires, puis l'unité elle-même.

Nous pouvons aussi noter que les dépendances de conflit et de remplacement n'interviennent pas. En effet, les dépendances de conflit ne sont pas déployées et permettent simplement de vérifier si une unité peut être déployée ou non. Les dépendances de remplacement, quant à elles, sont déployées si cela est nécessaire, mais pour remplacer une dépendance d'un autre type (par exemple une dépendance obligatoire). Dans ce cas l'ordre à prendre en compte est celui indiqué par l'autre type de dépendances.

Pour étendre ce type de stratégie au cas du déploiement de *n* applications sur *p* machines, la liste ordonnée peut être remplacée par une liste d'unités de déploiement. Ainsi, par exemple l'expression *ORDERED(U1, V7, W4)* signifie qu'il faut d'abord déployer l'unité *U1*, puis l'unité *V7*, et enfin l'unité *W4*.

4.3. Stratégies d'exécution

Des stratégies peuvent aussi être appliquées lors de l'exécution du plan de déploiement. Ces stratégies peuvent permettre d'affiner des choix d'unité de déploiement, s'il reste encore des libertés au niveau du plan de déploiement global. Elles servent aussi à traiter les cas d'erreurs possibles. Les différents types de stratégie de cette catégorie sont appliqués au moment opportun (quand un choix doit être fait, quand une erreur se produit, …).

4.3.1. Stratégies EXECUTION-SELECTION

Ces stratégies ont le même principe de fonctionnement que les stratégies *VERSION-DROITS* qui indiquent une préférence (*IF-AVAILABLE*). Dans ce cas, si plusieurs unités restent disponibles, le choix se porte vers celle qui remplit les conditions indiquées par l'expression logique. Elles peuvent alors exprimer une préférence pour des unité implémentées en java, pour des unités nécessitant peu d'espace disque, etc. Elles sont appliquées lorsque l'exécution de la structure du plan arrive sur un nœud où un choix est possible (nœud *XOR*, par exemple).

Comme le plan de déploiement global contient un ensemble de déploiements unitaires (concernant une machine), ce type de stratégie ne peut être appliqué que sur une machine.

D'autres types de stratégie d'exécution pourraient être définis pour exprimer comment doivent être résolus les différents degrés de liberté. Par exemple, un type *EXECUTION-CHRONOLOGIE* pourrait indiquer comment résoudre les ordres partiels (nœuds *AND* du plan) indiqués dans le plan de déploiement.

4.3.2. Stratégies EXECUTION-ERREUR

Les stratégies de type *EXECUTION-ERREUR* permettent de décider de ce qu'il faut faire en cas d'**erreur au cours de l'exécution du plan de déploiement** : arrêter, continuer le déploiement, etc. Il peut s'agir de tout type d'erreur lié à l'exécution des différentes étapes du plan. Par exemple, une erreur peut survenir lors d'un transfert de données. Plusieurs possibilités peuvent alors être indiquées par la valeur du champ *contrainte-Entreprise* de la stratégie :

- **STOP** : Le déploiement en cours est arrêté et la (les) machine(s) concernée(s) restent dans l'état où elle(s) est (sont).
- **CONTINUE** : Le déploiement se poursuit malgré le problème repéré.
- **ROLLBACK** : Le déploiement en cours est arrêté, et un retour en arrière des activités déjà effectuées est réalisé afin de ramener la (les) machine(s) concernée(s) dans son (leur) état initial.
- **ROLLBACK_CONTINUE** : Le déploiement en cours est arrêté, et un retour en arrière des activités déjà effectuées est réalisé sur la machine concernée. Le déploiement se poursuit ensuite par les déploiements sur les autres machines du groupe.

L'utilisation des stratégies *ROLLBACK* et *ROLLBACK_CONTINUE* sous-entend que le producteur met à disposition les activités permettant de réaliser ce retour en arrière.

Ce type de stratégies peut être utilisé au niveau des groupes et machines de l'entreprise. La stratégie doit être propagée jusqu'aux machines cibles. Les stratégies de ce type exprimées sur un groupe cachent les stratégies de même type appliquées aux sous-groupes et machines de ce groupe.

Certaines **erreurs sont dues à l'évolution de la cible depuis la construction du plan**. Nous pouvons citer les exemples suivants :

- les contraintes imposées par l'unité de déploiement ne sont plus vérifiées par la machine. Par exemple, l'espace disque disponible n'est plus suffisant pour accueillir l'unité.
- l'unité (ou une de ses dépendances) que l'on cherche à déployer est déjà déployée sur la cible.
- une machine cible s'est déconnectée.

Dans ce cas, d'autres types de stratégies plus élaborées peuvent être appliqués. Par exemple, en ce qui concerne la vérification des contraintes, une stratégie peut décider de « déployer à tout prix », c'est-à-dire de déployer malgré tout l'unité. Ceci peut impliquer des opérations sur le site, comme par exemple de libérer de l'espace disque. Une autre stratégie pourrait être de « déployer si possible ». Dans ce cas, si les contraintes ne sont pas vérifiées, le déploiement est abandonné.

Dans le cas où l'unité de déploiement a déjà été déployée entre la construction du plan de déploiement et son exécution, plusieurs choix sont aussi possibles. Par exemple, le plan de déploiement global peut être complètement abandonné pour en recalculer un nouveau, ou

alors, le déploiement unitaire de la machine concernée est considéré comme valide et le déploiement se poursuit sur les autres machines.

Dans le cas où une machine cible s'est déconnectée, il est possible d'abandonner le déploiement complet (et éventuellement de recalculer un nouveau plan de déploiement), d'attendre la reconnexion de la machine, ou de réaliser le déploiement sur les autres machines (encore connectées).

Nous ne définissons pas plus de types de stratégies ici. L'objectif de cette partie était de donner un ensemble d'exemples de ce qui peut être exprimé au moyen de stratégies de déploiement.

5. PRINCIPE D'APPLICATION DES STRATEGIES

Pour appliquer les stratégies, il faut parcourir la structure d'entreprise (cf. paragraphe 5.1 du chapitre 5) que nous avons définie dans le chapitre précédent. Ceci permet d'associer à chaque nœud la structure d'application (cf. paragraphe 5.2 du chapitre 5) qui convient en accord avec les stratégies de sélection.

Ensuite, un autre parcours de la structure d'entreprise permet de créer la structure de plan (cf. paragraphe 5.3 du chapitre 5) de déploiement en accord avec les stratégies d'ordonnancement (et les structures d'applications possibles).

Enfin, un parcours de la structure de plan permet l'exécution du plan de déploiement. Cette exécution peut mettre en jeu l'application de stratégies d'exécution.

Cette partie présente tout d'abord les algorithmes de parcours qui permettent d'appliquer les différentes stratégies. Puis, ces algorithmes sont illustrés par un exemple.

5.1. Algorithmes de parcours

Nous nous intéressons ici au parcours permettant d'effectuer les étapes :

- de sélection des unités à déployer, et,
- de création et d'ordonnancement du plan de déploiement global.

5.1.1. Sélection des unités à déployer

Pour débuter le déploiement, il faut sélectionner les unités qui sont possibles pour chacune des machines. Pour pouvoir effectuer cette étape, il faut utiliser les structures d'entreprise et d'application, qui sont construites en fonction de la demande de déploiement (cible et objet de déploiement). L'étape de sélection implique alors la mise en relation :

- des **contraintes** des unités disponibles avec les propriétés des machines cibles, et,
- des **stratégies** de groupes et de machines avec les propriétés des unités disponibles.

Une structure d'application, créée avec l'ensemble des unités disponibles, est transmise aux différents nœuds de la structure d'entreprise. A chaque nœud, des traitements peuvent être effectués. Ces traitements modifient la structure d'application, et correspondent à l'application des stratégies. Au niveau des machines, les contraintes des unités doivent aussi être vérifiées.

Le principe de l'algorithme est alors le suivant :

- **Etape 1** : Démarrer à la racine de la structure d'entreprise.
- **Etape 2** : Elaguer la structure d'application selon les instructions suivantes :
  - *Sur un nœud machine* : Vérifier les contraintes de chaque unité de la structure d'application (et supprimer les unités dont les contraintes ne sont pas vérifiées)
  - *Sur tout type de nœud* (machine et groupe) : Appliquer les stratégies de sélection, liées au nœud, sur la structure d'application.
- **Etape 3** : Exécuter les étapes 2 et 3 récursivement pour chaque fils du nœud, sur une copie de la structure d'application, jusqu'à atteindre un nœud machine.

Une fois cette étape réalisée, une structure d'application est associée à chaque nœud de la structure d'entreprise. Elle représente l'ensemble des unités possibles pour le nœud.

5.1.2. Création et ordonnancement du plan de déploiement

Une fois que toutes les unités de déploiement sont sélectionnées, on sait quelle unité ira sur quelle machine. En pratique, la structure d'entreprise contient l'information nécessaire pour construire graduellement la structure de plan. Au niveau des nœuds intermédiaires (groupes), des stratégies permettent d'établir un ordre de déploiement (partiel ou total). De plus, au niveau des nœuds machine, les couples <*unité, machine*> peuvent être créés à partir de la structure d'application associée au nœud. Il reste alors à ordonner l'ensemble de ces couples en respectant les stratégies de déploiement. L'ordonnancement de ces plans de déploiement unitaires permet alors de créer le plan de déploiement global.

A cette étape, il faut créer la structure de plan. Il s'agit de parcourir la structure d'entreprise (à laquelle sont associées différentes structures d'application, pour chaque machine) et de créer les nœuds de la structure de plan qui correspondent aux stratégies d'ordonnancement.

Le principe de l'algorithme est alors le suivant :

- **Etape 1** : Démarrer à la racine de la structure d'entreprise.
- **Etape 2** : Créer un nœud du plan de déploiement selon les instructions suivantes :
  - *Sur un nœud machine* : Créer le(s) couple(s) <unité, machine> selon la structure d'application. Si plusieurs couples sont nécessaires (déploiement de dépendances par exemple), les ordonner (et créer les nœuds du plan correspondants) selon les stratégies disponibles.
  - *Sur les autres nœuds* : Créer le nœud du plan qui convient en appliquant les stratégies d'ordonnancement, et l'ajoutant à la structure de plan.

- **Etape 3** : Exécuter les étapes 2 et 3 récursivement pour chaque fils du nœud, jusqu'à atteindre un nœud machine. Cette étape permet de construire récursivement les sous-arbres de la structure de plan.

Après cette étape, la structure de plan est disponible. Elle représente le plan de déploiement global qui permet de déployer un ensemble d'unités sur un groupe de machines (structure d'entreprise).

5.1.3. Exécution du plan de déploiement

Pour exécuter le plan de déploiement, il suffit de parcourir la structure de plan qui a été construite à l'étape précédente. Le parcours de la structure d'entreprise peut aussi permettre d'appliquer des stratégies d'exécution à différents moment de l'exécution : soit au moment où un choix se présente, soit quand une erreur survient, etc.

5.2. Exemple de déploiement avec stratégies de déploiement, sans dépendances

Reprenons l'exemple présenté dans la partie 7 du chapitre précédent. Nous cherchons à déployer l'application *U* disponible en 7 versions différentes (unités *U1*, *U2*, *U3*, *U4*, *U5*, *U6* et *U7*) sur les machines du groupe *Adèle*. Pour simplifier l'application des stratégies dans notre exemple, nous ne tenons pas compte des dépendances des unités. L'application des stratégies avec des unités qui ont des dépendances sera présentée dans la suite.

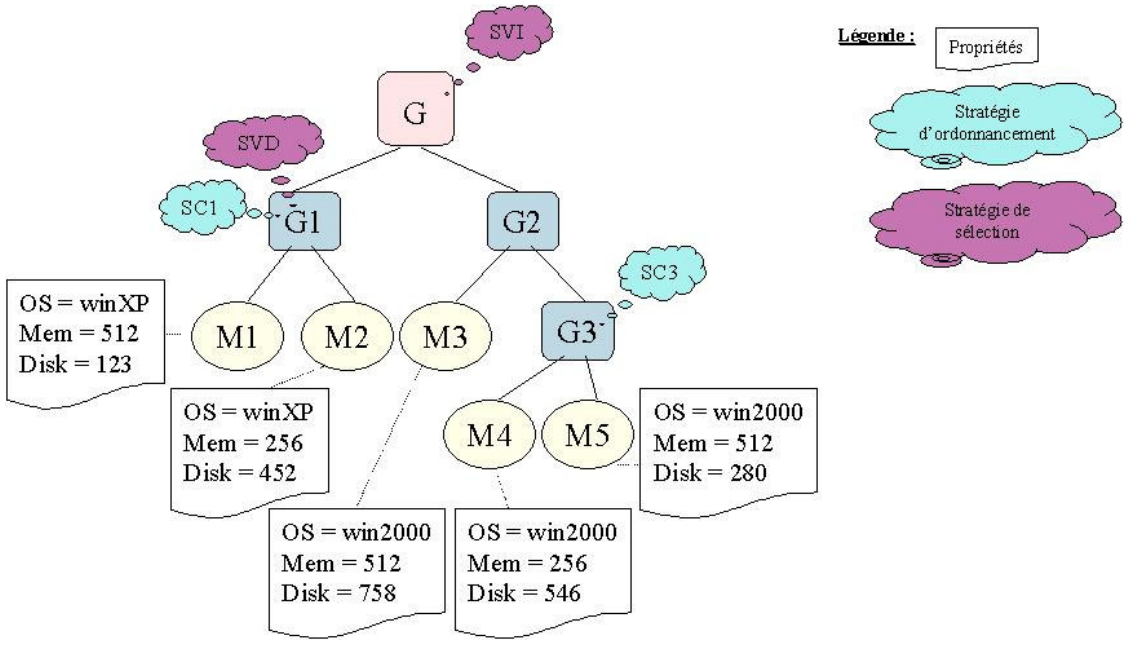

Figure 6.3 Structure d'entreprise appliquant des stratégies

Comme le montre la Figure 6.3, nous ajoutons des stratégies à la structure d'entreprise que nous avions dans le chapitre précédent :

- La stratégie *SVI*, appliquée au groupe *G*, est une stratégie de sélection de type *VERSION-INTRAGROUPE*. Elle est représentée par les champs < *INSTALL, 'Implémentation'=Java, UNIT* >. Cela signifie qu'il faut déployer la même unité, implémentée en *java*, sur toutes les machines du groupe *G*.
- La stratégie *SVD*, appliquée au groupe *G1*, est une stratégie de sélection de type *VERSION-DROITS*. Elle est définie par le triplet < *INSTALL, 'Interface'=multimedia, NEVER* >. Cela signifie qu'il ne faut pas déployer d'unité avec une interface multimédia sur les machines du groupe *G1*.
- La stratégie *SC1*, appliquée au groupe *G1*, est une stratégie d'ordonnancement de type *CHRONOLOGIE-MACHINES*. Elle est défini par le triplet < *INSTALL, 'Type'=devTool, PARALLEL(M1, M2)* >. Cela signifie qu'il faut déployer les outils de développement en parallèle sur les machines M1 et M2.
- La stratégie *SC3*, appliquée au groupe *G3*, est une stratégie d'ordonnancement de type *CHRONOLOGIE-MACHINES*. Elle est défini par le triplet < *INSTALL, 'Type'=devTool, ORDERED(M5, M4)* >. Cela signifie qu'il faut déployer les outils de développement d'abord sur la machine M5, puis sur la machine M4.

Les stratégies *SVI* et *SVD* sont donc utilisées pour la sélection des unités à déployer. Les stratégies *SC1* et *SC3* sont nécessaires pour l'étape de création et d'ordonnancement du plan de déploiement.

5.2.1. Sélection des unités à déployer

Pour effectuer la sélection des unités à déployer, il faut parcourir la structure d'entreprise en utilisant la structure d'application (Figure 5.18) et en tenant compte des stratégies à appliquer et des contraintes imposées par les unités.

Le parcours débute au niveau du nœud *G* où la stratégie *SVI* doit être appliquée. L'expression logique de la stratégie est utilisée comme une garde et ne conserve que l'*ensemble-Vrai* formé des unités implémentées en java, c'est-à-dire avec la propriété <*Implémentation, Java*> (*U2* est écartée). Le traitement de la stratégie est mis en attente jusqu'à ce que les informations en provenance des fils soient disponibles. Une copie de la structure d'application (avec les unités *U1*, *U3*, *U4*, *U5*, *U6*, *U7*) est transférée à chaque fils avec la stratégie *SV1* en attente sur le père.

Au niveau de *G1*, il faut appliquer la stratégie *SVD* et mettre en attente la stratégie « héritée » *SV1*. La stratégie *SVD* supprime toutes les unités avec une interface multimédia (c'est-à-dire *U6*). La nouvelle structure d'application (avec les unités *U1*, *U3*, *U4*, *U5*, *U7*) est transmise aux machines *M1* et *M2*. Au niveau des machines, la vérification des contraintes est effectuée (voir l'exemple dans le chapitre précédent).

Au niveau de *G2* la stratégie « héritée » *SV1* est mise en attente. Aucune stratégie de sélection n'est indiquée sur *G2*. Une copie de la structure d'application issue de *G* est alors transmise sans modifications à la machine *M3*. Cette même structure (une copie) est transmise à *G3* avec la stratégie *SV1* en attente. Sachant qu'il n'y a pas de stratégie de sélection sur *G3* la structure d'application (une copie) est transmise à *M4* et *M5*. A ces niveaux, la vérification des contraintes est aussi effectuée et permet d'élaguer la structure d'application en fonction des caractéristiques de chaque machine.

La Figure 6.4 est une étape intermédiaire de la sélection. Elle indique quelles unités sont possibles pour chaque machine, avant de finir d'appliquer la stratégie *SVI* (même unité pour toutes les machines). Les stratégies *SVI-like* représentent la propagation de la stratégie *SVI* au niveau des sous-groupes *G1*, *G2* et *G3*.

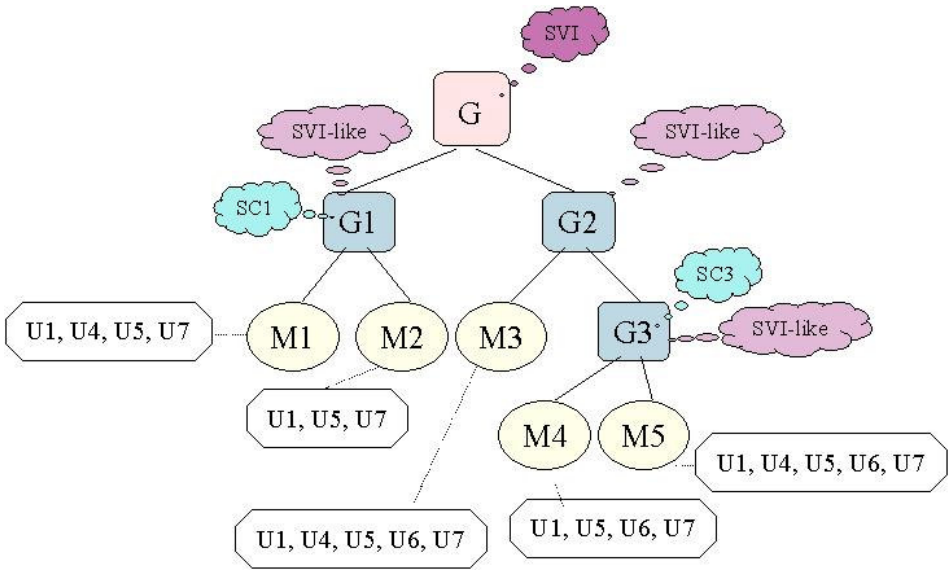

Figure 6.4 Unités possibles pour chaque machine

Une fois que l'information (unités possibles) est disponible pour chaque machine, la stratégie *SVI* peut être traitée. Il faut alors réaliser l'intersection des ensembles d'unités possibles pour chaque machine. Cette intersection est réalisée sur chaque nœud intermédiaire (*G3*, *G2* et *G1*) puis sur G, lors de la remontée dans la structure d'entreprise jusqu'à G. Ceci est un exemple de transmission de stratégie d'un nœud vers ses fils de type groupe. On obtient alors l'ensemble des unités qui sont en accord avec toutes les stratégies de sélection et dont les contraintes sont vérifiées par les caractéristiques de toutes les machines. Cet ensemble est composé des unités *U1*, *U5* et *U7*.

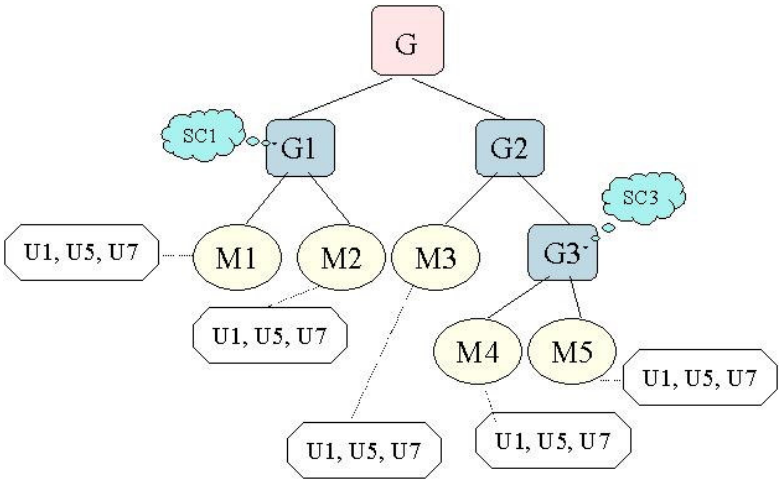

Figure 6.5 Résultat de la sélection des unités à déployer

La Figure 6.5 représente la structure d'entreprise (cible de déploiement), avec les unités possibles pour chaque machine et les stratégies d'ordonnancement qu'il reste à appliquer.

5.2.2. Création et ordonnancement du plan de déploiement

A l'issue de l'étape de sélection, trois unités sont encore possibles. Selon le souhait de l'entreprise, le choix de l'unité peut être effectué à ce moment (via une interaction de l'utilisateur ou via un tirage aléatoire parmi l'ensemble sélectionné, par exemple) ou il peut être reporté à plus tard et être intégré au plan de déploiement. Pour simplifier l'exemple, nous choisissons de déployer l'unité *U7*. L'interaction entre les différentes stratégies (ici, sélection et ordonnancement) est discutée plus tard.

La construction de la structure de plan est basée sur un autre parcours de la structure d'entreprise, en appliquant les stratégies d'ordonnancement. Le parcours débute au niveau de *G* où aucune stratégie d'ordonnancement n'est indiquée. Un nœud *and* (racine de la structure de plan) est donc créé. Ses fils sont créés récursivement en continuant le parcours de la structure d'entreprise.

Au niveau de *G1*, il faut appliquer la stratégie *SC1* qui indique que le déploiement sur les machines *M1* et *M2* doit être fait en parallèle. Un nœud *parallel* est donc créé. Ses fils sont les plans de déploiement unitaires qui correspondent au déploiement de l'unité *U7* sur les machines *M1* et *M2*.

Au niveau de *G2*, aucune stratégie n'est appliquée : un nœud *and* est créé. L'un de ses fils correspond au déploiement unitaire de *U7* sur *M3*, l'autre de ses fils correspond au déploiement sur le groupe *G3*.

Au niveau de *G3*, la stratégie *SC3* indique qu'il faut d'abord déployer sur *M5*, puis sur *M4*. Un nœud *ordered* est donc créé avec pour fils ces deux déploiements unitaires.

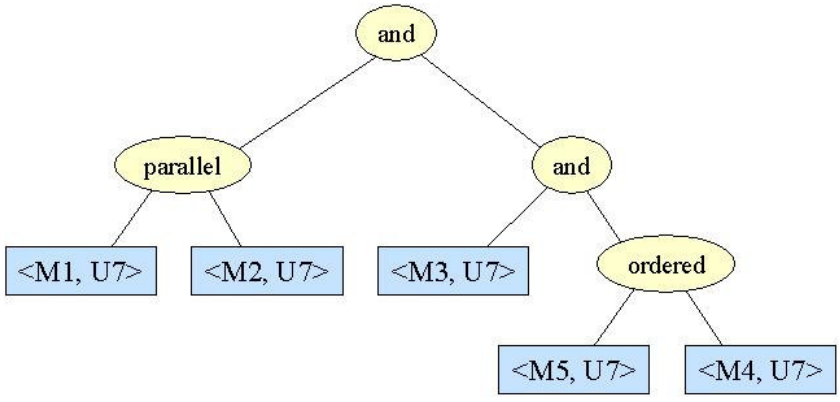

Figure 6.6 Structure de plan

La Figure 6.6 présente la structure de plan qui est obtenue en appliquant les stratégies d'ordonnancement *SC1* et *SC3*. Il faut noter que lors de l'exécution de ce plan de déploiement, des choix d'ordonnancement peuvent encore être faits sur les nœuds *and*. Ces choix peuvent être orientés par des stratégies d'exécution.

6. GENERALISATION : DEFINITION D'UN LANGAGE DE DESCRIPTION DE STRATEGIES

Les exemples que nous avons présentés correspondent aux stratégies que nous avons identifiées pour répondre aux principaux problèmes principaux de déploiement que nous avons rencontrés lors de collaborations industrielles. Pour plus de flexibilité et pour pouvoir définir n'importe quel type de stratégie, nous proposons un langage de stratégie. Pour cela, nous décrivons la grammaire BNF correspondante, puis nous donnons quelques exemples permettant de la valider.

6.1. Grammaire du langage de stratégies

Une stratégie possède des caractéristiques qui sont définies au niveau de son type. Nous commençons donc par définir ce que représente un type de stratégie.

6.1.1. Type de stratégie

Un type de stratégie est représenté par son nom (*nomType*) qui est un simple identificateur, sa catégorie, l'ensemble de ses caractéristiques (*position*, *visibilité*, *propagation* et *précédence*) et son interprétation.

```
< typeStratégie > ::= < nomType, catégorie, position,
      visibilité, propagation, précédence, interprétation>
```

L'interprétation de la stratégie correspond à la manière d'appliquer la stratégie et n'est pas définie ici : il s'agit d'écrire la méthode qui permet d'appliquer la stratégie en fonction des contraintes d'unités et d'entreprise.

Les stratégies de déploiement peuvent se décomposer en deux niveaux :

- Le **niveau Déclaration** définit les contraintes imposées par l'entreprise. Il représente l'expression de la sémantique globale de la stratégie à traiter.
- Le **niveau Méthode** détermine comment appliquer la stratégie. Plusieurs méthodes peuvent être possibles pour appliquer la même stratégie. Ce niveau représente le traitement de la sémantique indiquée au niveau déclaration de stratégie.

Par exemple, au niveau stratégie, l'entreprise décide de déployer la même version d'une unité sur un groupe de machines. Au niveau méthode, il s'agit de choisir une version qui soit compatible avec les configurations de toutes les machines du groupe ou de décider d'une version (selon certains critères) et d'imposer que toutes les machines aient une configuration compatible avec cette version.

La catégorie du type de la stratégie correspond au moment où la stratégie doit être appliquée. On retrouve donc les catégories de stratégies que nous avons définies précédemment : sélection, ordonnancement, exécution. D'autre catégories peuvent éventuellement être ajoutés par l'utilisateur. Par exemple, une catégorie peut regrouper des stratégies qui concernent le moment d'exécution du plan de déploiement (planification de déploiement). Un autre type de stratégie peut aussi être utilisé pour optimiser le déploiement (le plus de machines cibles, le moins de temps possible (disponibilité du réseau), etc.).

```
< catégorie > ::= SELECTION | ORDONNANCEMENT | EXECUTION
```

Il reste ensuite à définir les différentes caractéristiques du type de la stratégie que nous avons définies précédemment. Nous pouvons noter que la portée n'apparaît pas ici puisqu'elle est commune à tous les types de stratégie. En effet, une stratégie appliquée à un groupe doit s'appliquer à toutes les machines du groupe, alors qu'une stratégie appliquée à une machine ne concerne que cette machine.

```
< position > ::= GROUPE | MACHINE | ENTITE
```

La position définit à quelle type d'entité de l'entreprise peut s'appliquer ce type de stratégie : soit aux groupes (*GROUPE*), soit aux machines (*MACHINE*), soit aux deux (*ENTITE*).

```
< visibilité> ::= MASQUER | PAS-MASQUER
```

La visibilité détermine si la stratégie doit cacher (*MASQUER*) ou non (*PAS-MASQUER*) les stratégies de même type des niveaux inférieurs de la structure d'entreprise (sous-groupes et machines).

```
< propagation> ::= NON | FILS | GROUPE | MACHINE
```

La propagation détermine si la stratégie doit ou non être propagée aux niveaux inférieurs de la structure d'entreprise. La stratégie peut être propagée à tous les fils (*FILS*) quelque soit leur type ou à aucun d'eux (NON). Elle peut aussi n'être propagée qu'au niveau des sous-groupes (*GROUPE*) ou des fils de type machine (*MACHINE*).

```
< précédence> ::= HERITEES | LOCALES
```

La précédence détermine la priorité qu'il existe entre les stratégies de même type au niveau d'un même nœud. La priorité peut être accordée aux stratégies héritées (*HERITEES*), c'est-à-dire appliquées au niveau d'un groupe père, ou aux stratégies locales (*LOCALES*), appliquées sur le nœud. Les stratégies globales ne sont pas mentionnées, puisqu'elles peuvent être vues comme des stratégies héritées à partir de la racine de la structure d'entreprise.

Avec cette partie du langage, l'utilisateur peut définir les types de stratégie qui lui conviennent le mieux et la manière de les appliquer. Après avoir défini ce qu'est un type de stratégie, il reste à définir comment représenter une stratégie.

6.1.2. Expression de stratégie

Nous avons vu, dès le début de ce chapitre, qu'une stratégie est caractérisée par :

- l'activité de déploiement pour laquelle elle est définie,
- l'expression d'une contrainte sur les unités, c'est-à-dire sur quelle unités doit s'appliquer la stratégie,
- l'expression d'une contrainte sur l'entreprise, c'est à dire ce qu'impose la stratégie.

Nous rajoutons ici le type de la stratégie (*nomType*) qui permet de lui associer les caractéristiques qui viennent d'être énoncées.

```
< stratégie> ::= <activitéDeDéploiement {, contrainte-Unités},
```

```
contrainte-Entreprise, nomType>
```

L'activité de déploiement (*activitéDeDéploiement*) est le nom de l'une des activités du cycle de vie du déploiement. Il s'agit de déterminer pour quelle activité de déploiement s'applique la stratégie.

```
< activitéDeDéploiement> ::= INSTALL | UPDATE | RECONFIGURE |
            UNINSTALL | ACTIVATE | DESACTIVATE
```

La contrainte sur les unités (*contrainte-Unités*) est une expression booléenne qui fait intervenir des noms de propriétés des unités et leurs valeurs possibles, des opérateurs logiques et/ou opérateurs ensemblistes classiques. Cette contrainte est optionnelle : dans le cas où aucune contrainte n'est spécifiée, cela signifie que la stratégie s'applique pour tous les types d'unités. Cela revient à remplacer l'évaluation de la contrainte par la valeur *vrai*.

La contrainte sur l'entreprise (*contrainte-Entreprise*) détermine la contrainte imposée par l'entreprise. Elle peut être exprimée de différentes manières, faisant, par exemple, intervenir des mots-clés, des noms de machines, etc. C'est une partie du langage qui doit être complété par l'utilisateur en fonction des types de stratégies qu'il définit et de l'information nécessaire pour les interpréter.

6.2. Quelques exemples

Pour définir une nouvelle stratégie, il faut :

- définir le type de la stratégie et ses caractéristiques,
- étendre le langage pour déterminer comment exprimer la contrainte d'entreprise,
- définir la méthode qui permet d'interpréter la stratégie (champs *contrainte-Unités* et *contrainte-Entreprise*), et,
- exprimer la stratégie avec le langage.

Pour donner un exemple, nous définissons les types de stratégies que nous avons appelés *VERSION-DROITS* et *CHRONOLOGIE-MACHINES*.

6.2.1. Définition des types de stratégie

Deux types de stratégies sont donc définis *VERSION-DROITS* et *CHRONOLOGIE-MACHINES*, par les exemples 1 et 2.

Exemple 5 : Type de stratégie VERSION-DROITS

```
< VERSION-DROITS, SELECTION, ENTITE, PAS-MASQUER, NON,
LOCALES, interprétationVdroits>
```

Une stratégie *VERSION-DROITS* est une stratégie de sélection (*SELECTION*) qui peut s'appliquer à un groupe ou une machine (*ENTITE*). Au niveau visibilité, elle ne masque pas (*PAS-MASQUER*) les stratégies de même type des niveaux inférieurs de la structure d'entreprise. Elle n'est pas propagée (*NON*) et la priorité est accordée aux stratégies locales (*LOCALES*). En fait, la priorité n'est pas utilisée puisque chaque stratégie de ce type est

traitée au niveau du nœud où elle s'applique. Enfin, ce type de stratégie est interprété par la méthode *interprétationVdroits*.

Exemple 6 : Type de stratégie CHRONOLOGIE-MACHINES
< CHRONOLOGIE-MACHINES, ORDONNANCEMENT, GROUPE, PAS-MASQUER, NON, HERITEES, interprétationChrono >

Une stratégie *CHRONOLOGIE-MACHINES* est une stratégie d'ordonnancement (*ORDONNANCEMENT*) qui ne peut s'appliquer qu'à un groupe (*GROUPE*). Au niveau visibilité, elle ne masque pas (*PAS-MASQUER*) les stratégies de même type des niveaux inférieurs de la structure d'entreprise. Elle n'est pas propagée (*NON*) et la priorité est accordée aux stratégies héritées (*HERITEES*). Enfin, ce type de stratégie est interprété par la méthode *interprétationChrono*.

6.2.2. Extension du langage de stratégies

Pour pouvoir exprimer la contrainte d'entreprise (*contrainte-Entreprise*), il faut étendre le langage de stratégie avec les mots-clés et expressions qui conviennent.

Exemple 7 : Extension du langage de stratégies
< **contrainte-Entreprise** > ::= expressionVdroits |
expressionChronoMach
< **expressionVdroits** > ::= NEVER | ALWAYS | SI-POSSIBLE
< **expressionChronoMach** > ::= ordre(listeMachinesOuGroupes) |
ordre(entier)
< **ordre** > ::= ORDERED | PARALLEL | AND | …
< **ListeMachinesOuGroupes** > ::= 'identificateur' |
'identificateur', ListeMachinesOUGroupes

Une expression de stratégie *VERSION-DROITS* (*expressionVdroits*) peut être l'un des mots-clés qui ont été présentés au paragraphe 4.1.1 (*NEVER*, *ALWAYS* ou *SI-POSSIBLE*).

Une expression de stratégie *CHRONOLOGIE-MACHINES* (*expressionChronoMach*) est une liste de machines et/ou groupes ordonnée selon un mot-clé, ou un entier. Ces ordres ne peuvent pas être imbriqués. En effet, la stratégie concerne toutes les machines et sous-groupes du nœud auquel est accroché la stratégie. Ainsi, soit tous les fils du nœud sont énumérés dans la liste *listeMachinesOuGroupes*, soit un entier indique le nombre de fils sur lesquels doivent être déployés l'unité.

6.2.3. Définition des méthodes d'interprétation

Pour pouvoir utiliser un nouveau type de stratégie, il faut définir la méthode qui permet de l'interpréter et de traiter les contraintes sur les unités et l'entreprise. Nous définissons ici brièvement une structure d'algorithme qui permet d'interpréter les deux types de stratégies définis dans les exemples.

Exemple 8 : Interprétation des stratégies de type *VERSION-DROITS*
InterprétationVdroits() :
EnsVrai <- UnitésVérifContrainte-Unités(UnitésPossibles)
EnsFaux <- UnitésNonVérifContrainte-Unités(UnitésPossibles)
Selon contrainte-Entreprise :
NEVER : choisir EnsFaux

ALWAYS : choisir EnsVrai
IF-AVAILABLE : si EnsVrai ≠ vide
alors choisir EnsVrai
sinon choisir EnsFaux

L'interprétation des stratégies *VERSION-DROITS* consiste à séparer l'*ensemble-Vrai* et l'*ensemble-Faux*. Selon le mot-clé utilisé dans la *contrainte-Entreprise*, c'est l'un ou l'autre de ces deux ensembles qui est sélectionné.

Exemple 9 : Interprétation des stratégies de type *CHRONOLOGIE-MACHINES*

```
InterprétationChrono() :
        Si Verifier(UnitésPossibles, contrainte-Unités)
        Alors créerNoeudPlan (ordre)
                EffectuerAppelRecursifPourConstruireFils
                                        (ListeMachinesOuGroupes)
```

Les stratégies *CHRONOLOGIE-MACHINES* ne sont appliquées que pour les unités qui vérifient la *contrainte-Unités*. Ces stratégies sont utilisées pour créer et ordonner le plan de déploiement global. Il s'agit donc de créer la structure qui correspond à l'expression de la *contrainte-Entreprise*.

6.2.4. Expression des stratégies

Il ne reste plus qu'à exprimer les stratégies qui répondent aux définitions que nous venons de réaliser.

Exemple 10 : Expression d'une stratégie *VERSION-DROITS*

< INSTALL, 'Interface' = multimedia, NEVER, VERSION-DROITS >

Cette stratégie de type *VERSION-DROITS* indique qu'il ne faut pas installer d'unités ayant la propriété <*Interface, multimedia*>.

Exemple 11 : Expression d'une stratégie *CHRONOLOGIE-MACHINES*

< INSTALL, ORDERED('M14', 'M7'), CHRONOLOGIE-MACHINES >

Cette stratégie de type *CHRONOLOGIE-MACHINES* indique qu'il faut installer les unités de déploiement d'abord sur *M14*, puis sur *M7*. Comme aucune *contrainte-Unités* n'est exprimée, cette stratégie concerne toutes les installations.

7. INTERACTION DES STRATEGIES

Les paragraphes précédents ont décrit comment définir et appliquer des stratégies. Les exemples restent assez simples car une seule stratégie (ou une stratégie de chaque catégorie) est utilisée. Cependant, les cas d'étude se compliquent lorsque plusieurs stratégies interviennent. Dans cette partie, nous étudions différentes interactions qui peuvent exister. Cette partie présente donc quelques problèmes que nous avons pu identifier au cours de nos expériences, mais qui n'ont pas été résolus dans le cas général. Ces problèmes devront être traités dans les versions de notre environnement ORYA, qui seront opérationnelles pour supporter des stratégies élaborées.

7.1. Interaction des stratégies entre elles

Tout d'abord, plusieurs stratégies peuvent intervenir pour une même unité et une même cible. Deux types d'interaction peuvent alors être identifiés :

- la composition de stratégie, et,
- le conflit entre stratégies.

7.1.1. Composition de stratégies

Pour simplifier notre exemple, nous avons décidé de définir quelle unité déployer à la fin de l'étape de sélection. Cependant, l'entreprise peut aussi choisir de n'effectuer ce choix que le plus tard possible (au moment de l'exécution du plan de déploiement). Dans notre exemple, les unités U1, U5 et U7 étaient alors encore possibles (Figure 6.5). Dans ce cas, le plan de déploiement à construire est différent.

Lorsque les différents plans unitaires sont à ordonnancer pour produire le plan de déploiement global, il faut alors savoir quelles relations existent entre les plans unitaires. En effet, selon elles, des plans de déploiement différents peuvent être construits. La Figure 6.7 présente deux cas possibles. Pour simplifier les schémas, la figure réduit l'exemple précédent au déploiement de *U* (les unités *U1*, *U5* et *U7* sont possibles) sur le sous-groupe *G1*.

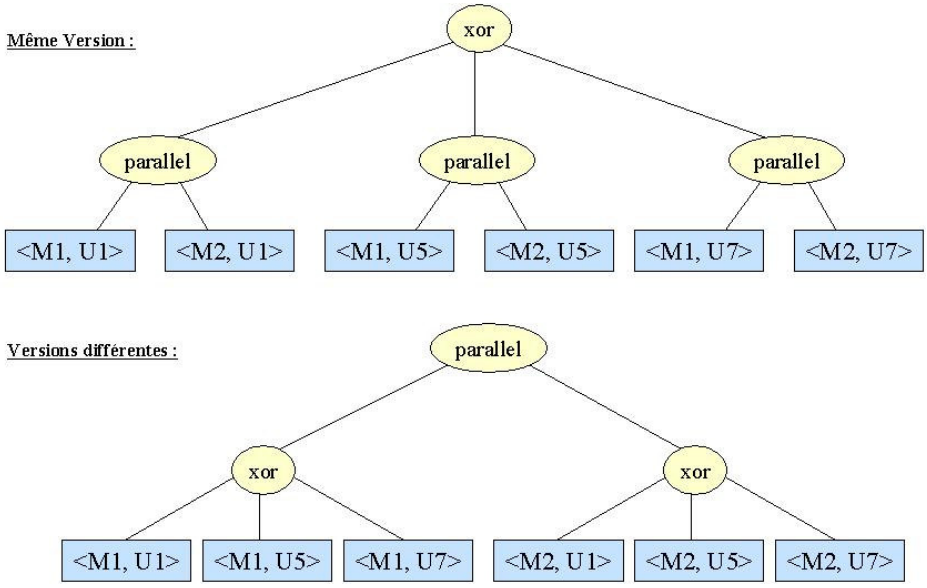

Figure 6.7 Composition de stratégies : quel plan de déploiement construire?

Ainsi, il faut, par exemple, savoir si :

- La même version doit être déployée sur toutes les machines (plan du haut de la Figure 6.7) : la racine de la structure de plan est un nœud *xor*. Les sous-arbres fils permettent alors de déployer la même unité sur plusieurs machines.
- Une version différente peut être déployée (plan du bas de la Figure 6.7) : la racine de la structure de plan est un nœud qui force de déploiement de tous les

sous-arbres (noeuds *and* ou *parallel* par exemple). Chaque arbre fils permet ensuite le déploiement d'une unité sur une machine (utilisation de nœuds *xor*).

Plusieurs catégories de stratégies doivent alors être composées entre elles pour définir quel plan de déploiement construire. Dans l'exemple que nous venons de citer, l'application des stratégies d'ordonnancement nécessite la consultation des stratégies de sélection (qui ont déjà été appliquées à l'étape précédente).

Une autre solution pourrait être de ne parcourir qu'une seule fois la structure d'entreprise. Cet unique parcours permettrait à la fois de sélectionner les unités à déployer et d'ordonnancer le plan de déploiement, en utilisant les deux types de stratégies. Cette étape reste cependant très complexe à réaliser de manière automatisée, puisque les deux types de stratégies peuvent interagir et modifier la sélection et/ou l'ordonnancement.

7.1.2. Conflit entre stratégies

Plusieurs stratégies de même type peuvent être applicables et contradictoires au moment du déploiement. Ainsi, par exemple, les stratégies de *CHRONOLOGIE-MACHINES <INSTALL, 'Implémentation' = Java, ORDERED(Machine1, Machine2), CHRONOLOGIE-MACHINES>* et *<INSTALL, 'Catégorie' = OutilDeDéploiement, ORDERED(Machine2, Machine1), CHRONOLOGIE-MACHINES>* sont définies pour un groupe de machines. Dans le cas d'une installation d'une unité de déploiement ayant les deux propriétés *<Catégorie, OutilDeDéploiement>* et *<Implémentation, Java>*, les deux ordres de déploiement indiqués par les stratégies sont en conflit.

Pour résoudre les problèmes de conflit entre stratégies qui peuvent apparaître, dans le cas où plusieurs sont applicables, des règles doivent être établies. Ces règles permettent à l'utilisateur de déterminer quelle(s) stratégie(s) doivent être prioritaires par rapport aux autres. Pour ce faire, il classe les stratégies définies par ordre croissant d'importance.

Dans ce genre de cas, comme toutes les stratégies ne peuvent pas être appliquées, l'utilisateur est informé non seulement des stratégies qui sont utilisées, mais aussi des stratégies qui n'ont pas pu être appliquées. De plus, si aucune règle de précédence n'est définie et que le conflit ne peut pas être résolu, l'utilisateur devra alors choisir interactivement quelle stratégie appliquer parmi celles possibles. A ce moment, l'utilisateur peut aussi choisir d'abandonner le déploiement, s'il considère que toutes les conditions nécessaires ne peuvent pas être remplies.

Pour éviter les conflits possibles, il faut aussi pouvoir les repérer. Cela signifie qu'avant l'application des stratégies, toutes les stratégies applicables (selon la contrainte sur les unités) de même type doivent être comparées pour vérifier si leurs contraintes sur l'entreprise ne sont pas contradictoires. Quelques cas assez simples peuvent être identifiés pour repérer automatiquement les conflits possibles. Cependant, pour certains types de stratégies utilisés lors de la sélection, les cas peuvent être plus complexes et demander une étude plus détaillée.

7.2. Interaction des stratégies avec les dépendances

Lors d'un déploiement d'unités avec dépendances, il faut déterminer quelle(s) stratégie(s) appliquer aux dépendances. Les cas suivants sont possibles :

- Les mêmes stratégies sont utilisées pour les unités à déployer et leurs dépendances. Dans ce cas, les unités sont considérées comme une partie de

l'unité à déployer et les stratégies de l'unité sont aussi appliquées à ses dépendances. Par exemple, si le déploiement de l'unité interdit une implémentation en java, alors, les unités dont des dépendances sont implémentées en java sont supprimées de l'ensemble des possibilités.

- Des stratégies différentes sont utilisées pour les unités et leurs dépendances. Dans ce cas, les dépendances sont considérées comme des unités à part entière, liées à leurs propres stratégies. Elle seront alors traitées comme des unités standards. Dans l'exemple précédent, les unités non implémentées en java mais ayant des dépendances implémentées en java peuvent être conservées parmi les possibilités.
- Les ensembles *ensemble-Vrai* et *ensemble-Faux* sont déterminés en fonction des propriétés des unités et de leurs dépendances. Ainsi, seules les unités, dont les propriétés remplissent la *contrainte-Unités* et dont les dépendances remplissent aussi cette contrainte, font partie de l'*ensemble-Vrai*. Les unités qui remplissent la *contrainte-Unités*, mais dont l'une des dépendances ne la vérifie pas, font partie de l'*ensemble-Faux*.

Lors de l'interprétation des stratégies, il faut déterminer lequel de ces cas appliquer.

8. EVALUATION DES CHOIX

Nous avons dû réaliser plusieurs choix (classification, définition, interaction, …) pour définir et mettre en œuvre des stratégies de déploiement. Ceux-ci présentent des avantages et des inconvénients.

8.1. Interaction des stratégies

L'un des avantages des stratégies est de laisser beaucoup de possibilités à l'utilisateur. En effet, grâce à la définition d'un langage de description de stratégie, l'utilisateur peut définir lui-même les types de stratégies qui l'intéressent.

Cependant, peut-être trop de libertés sont laissées, puisqu'il est difficile de gérer les différentes interactions qui peuvent exister lors de l'application des stratégies. Il est, en effet, difficile de savoir quelles stratégies sont appliquées lors d'un déploiement, parmi toutes celles qui ont été définies. Il serait envisageable de définir une priorité entre les différentes stratégies à appliquer. Mais le cas reste sensiblement le même lorsque deux stratégies de même priorité sont applicables. Il s'agit ici des mêmes inconvénients qui ont pu être identifiés dans le cadre de la programmation orientée aspect [KLM+97].

La difficulté consiste alors à trouver un équilibre, permettant à la fois de contrôler les stratégies appliquées et d'offrir un ensemble de possibilités assez large. Pour contrôler davantage l'application des stratégies, il est possible de restreindre un peu les possibilités offertes pour limiter les interactions avec les stratégies, notamment :

- en forçant les *contrainte-Unités* à être exclusives entre elles, ou en instaurant des priorités strictes entre elles (pour éliminer les conflits entre stratégies de même type),

- en forçant une sélection complète des unités à déployer avant de débuter l'ordonnancement du plan de déploiement (pour éviter la composition de stratégies de catégories différentes),
- en imposant un choix sur la manière d'utiliser les stratégies pour les dépendances des unités.

Il serait alors aussi possible d'imaginer des méta-stratégies permettant de déterminer quelles stratégies appliquer et comment résoudre les interactions entre elles. Selon les déploiements, il deviendrait possible d'utiliser les stratégies différemment. Par exemple, une méta-stratégie peut indiquer quelles stratégies appliquer pour les dépendances d'une unité : celles appliquées à l'unité ou celles spécifiques à chaque dépendance. De la même manière, il est possible d'indiquer que la sélection doit être terminée avant de commencer l'ordonnancement, ou que certaines stratégies de sélection peuvent encore être consultées lors de l'ordonnancement.

8.2. Structure de plan et structure d'entreprise

Des choix ont aussi été effectués au niveau du plan de déploiement. Nous avons vu que des degrés de liberté peuvent encore être présents dans le plan de déploiement global, au moment où son exécution débute. Cependant, nous avons restreint les choix par un nœud *xor* dans la structure de plan. Pour simplifier la sélection de l'un des plans de déploiement unitaires (couple <*machine, unité*>), nous avons forcé la sélection des unités à la fin de l'étape de sélection. Ceci implique que, sur un nœud *xor*, le choix doit être fait en fonction des caractéristiques des machines (et non des unités). Les seuls degrés de liberté accordés au plan de déploiement concernent alors l'ordonnancement, et de choix de cibles.

Pour que les degrés de liberté concernent aussi la sélection des unités, il faudrait définir un nouveau type de nœud dans la structure de plan. Ces nœuds permettraient d'exprimer le choix à faire, et d'indiquer s'il doit être fait sur les unités de déploiement ou sur les machines cibles.

Nous pourrions aussi imaginer que la structure d'entreprise est construite à partir de stratégies. Elle serait alors exactement la structure sur laquelle le déploiement doit être effectué. Ceci permettrait alors de supprimer les choix de machines, puisque chaque machine appartenant à la structure devra faire partie du déploiement : le choix aura été effectué lors de la construction de la structure d'entreprise. Un telle construction permet aussi de ne considérer que les groupes et sous-groupes de machines nécessaires à un déploiement particulier, et de leur appliquer les stratégies souhaitées. Ces stratégies pourraient alors être sélectionnées, lors de chaque nouveau déploiement, parmi une liste de l'entreprise.

9. SYNTHESE

Dans ce chapitre, nous avons décrit comment définir et mettre en œuvre des stratégies de déploiement, dans le cadre du déploiement à grande échelle pour les entreprises. Une stratégie permet de personnaliser une activité de déploiement, en fonction d'une contrainte imposée sur les unités à déployer (*contrainte-Unités*), et d'une contrainte imposée sur l'entreprise (*contrainte-Entreprise*).

Les stratégies ont aussi été classées en différentes catégories (sélection, ordonnancement, exécution). Chaque catégorie contient plusieurs types de stratégies, qui sont définis par leur position, leur portée, leur visibilité, leur propagation et leur précédence. Nous n'avons

présenté que quelques types de stratégies comme par exemple, les stratégies *VERSION-DROITS*, les stratégies *VERSION-INTRAGROUPE*, les stratégies *CHRONOLOGIE-MACHINES*, les stratégies *EXECUTION-ERREUR*, etc.

Nous avons ensuite exprimé le principe de mise en oeuvre des stratégies. Ce principe utilise les structure d'entreprise et d'application que nous avions définies dans le chapitre précédent. L'application de toutes les stratégies permet de construire une structure de plan. Cette structure représente le squelette du plan de déploiement global, qui permet de déployer un ensemble d'unités sur un ensemble de machines.

Pour donner plus de possibilités à l'utilisateur, et pour ne pas se limiter aux types de stratégies que nous avons identifiés, nous avons aussi défini un langage de description de stratégies. Celui-ci permet tout d'abord de définir les caractéristiques de chaque type de stratégies. Il permet ensuite de définir des stratégies ayant les types ainsi définis..

Enfin, nous avons terminé par une discussion sur les interactions qui peuvent exister entre les stratégies, ou entre les stratégies et les dépendances des unités. Cette discussion se termine par ce qui peut être considéré comme le travail à poursuivre pour définir des méta-stratégies. Des méta-stratégies permettraient alors de spécifier les interactions des stratégies en définissant précisément comment les résoudre.

CHAPITRE 7 : ORYA : REALISATION ET EVALUATION

Préambule :

Les chapitres précédents ont défini les concepts à utiliser et leur mise en oeuvre pour fournir un environnement automatisé, pour le déploiement d'applications à grande échelle, utilisant des stratégies. Ce chapitre présente la réalisation qui a été effectuée en deux étapes (environnement automatisé, puis orienté stratégies) et son évaluation.

---

1. INTRODUCTION

Le chapitre 5 a présenté la mise en œuvre des modèles, en utilisant les technologies de fédérations et de procédés. Dans ce chapitre, nous introduisons la réalisation de ce qui a été décrit précédemment. Pour cela, nous présentons notre environnement de déploiement : ORYA.

Comme pour les concepts qui ont été décrits, nous avons réalisé l'implémentation en deux étapes :

- La **première version d'ORYA** se focalise sur l'automatisation du déploiement. Elle permet de réaliser le déploiement automatisé d'une version d'une application sur une machine. Elle ne prend pas en compte les informations issues de l'entreprise.
- La **deuxième version d'ORYA** met l'accent sur l'environnement de l'entreprise. Elle permet le déploiement automatisé d'une application (éventuellement, plusieurs versions) sur un groupe de machines. Cette version d'ORYA prend en compte le niveau entreprise du déploiement et les stratégies de déploiement.

Dans ce chapitre, nous présentons tout d'abord la réalisation de la première version d'ORYA. Cette version a été expérimentée dans un milieu industriel, ce qui a permis d'évaluer les points forts et points faibles de cette première approche. Puis, nous abordons la réalisation de la deuxième version d'ORYA, suivie de son évaluation. C'est cette deuxième version qui constitue la partie la plus originale de mon travail d'implémentation

2. UN ENVIRONNEMENT POUR LE DEPLOIEMENT AUTOMATISE : ORYA – VERSION 1

Nous définissons ici une première version de notre environnement de déploiement ORYA (Open enviRonment to deploY Applications). Parmi les concepts de base [MB04-a], seuls les modèles de produit et de site sont utilisés dans cette première version. Le modèle d'entreprise et les stratégies sont utilisés dans la deuxième version de notre environnement.

Une version d'une application (unité de déploiement) est décrite par un manifeste, qui contient l'ensemble de l'information sur l'application : caractéristiques, contraintes, etc. Une machine est décrite par son descripteur (modèle) de site. Chaque machine, considérée comme client de la fédération, peut être un serveur d'applications et/ou un site client. De plus, un serveur de déploiement (machine où s'exécute le moteur de la fédération) déclenche et contrôle le déploiement d'applications.

Un serveur d'applications est chargé de fournir un ensemble de versions d'applications (sous forme packagée) à déployer et l'information qui les concerne. Un package contient toute l'information nécessaire au déploiement de l'application, à savoir :

- les **données** nécessaires au déploiement (fichiers à exécuter ou à copier, par exemple),

- le **manifeste** de l'application (modèle d'application), et,
- la description du **procédé** (« mode d'emploi ») de déploiement spécifique (qualifié de déploiement unitaire) à l'application.

Enfin, les sites clients, eux, sont les sites où sont déployées les applications. Un site est caractérisé par ses caractéristiques matérielles (mémoire, processeur,…) et logicielles (système d'exploitation, …). Il gère un ensemble d'applications qui y sont déployées.

2.1. Procédé de déploiement

Pour cette implémentation et cette expérimentation, nous nous focalisons sur l'activité d'installation. Pour cela, nous définissons un procédé (le plus générique possible) décrivant les étapes à franchir pour installer une application. Ce procédé représente un plan de déploiement unitaire, permettant d'installer une application sur une machine.

La Figure 7.1 décrit le procédé d'installation dans le langage APEL [DEA98]. Les rectangles représentent les activités, qui peuvent être simples ou composites (c'est-à-dire composées de plusieurs sous-activités). Chaque activité à des ports d'entrée et de sortie, par lesquels les produits transitent. Enfin, les produits sont transmis entre les différentes activités par les flux de données.

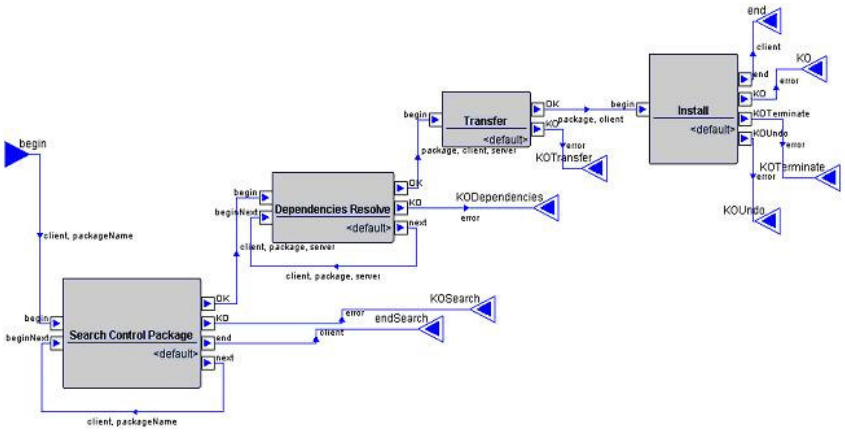

Figure 7.1 Procédé d'installation

Selon la définition de ce procédé [MB04-b], pour réaliser une installation, il faut commencer par rechercher le package de l'application souhaitée (activité *Search Control Package*) sur un serveur d'applications qui la fournit. Ensuite, il faut résoudre les dépendances éventuelles (activité *Dependencies Resolve*). Dans le cas où il y a des dépendances à résoudre, un autre procédé d'installation est lancé.

L'activité suivante réalise le transfert des données du serveur d'applications vers le site client (activité *Transfer*). Dans notre cas, comme nous utilisons la technologie des fédérations et que les machines ne se connaissent pas entre elles, le transfert est réalisé en deux étapes : du serveur d'applications vers un serveur de déploiement, puis du serveur de déploiement vers le site client. Nous rappelons qu'un serveur de déploiement est un serveur depuis lequel le déploiement est géré et surveillé. Dans l'implémentation, ce serveur correspond à la machine

sur laquelle le moteur de la fédération s'exécute. Enfin, l'installation physique de l'application sur le site client est réalisée.

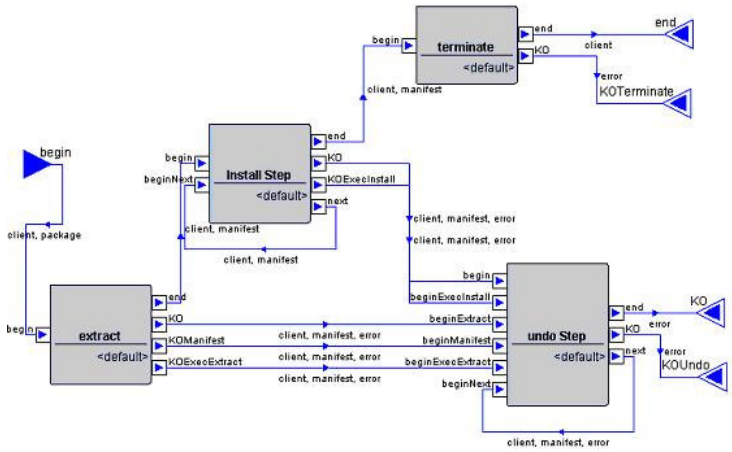

Figure 7.2 Activité d'installation

L'activité d'installation physique (*Install*) est une activité composite qui se décompose selon la Figure 7.2. La première activité consiste à extraire toutes les données du package. Ainsi, il est possible d'accéder au descripteur de déploiement qui décrit les activités d'installation spécifiques à l'application.

L'activité d'extraction (*Extract*) est suivie d'un ensemble d'étapes d'installation (*Install Step*), décrites dans le descripteur de déploiement. Ces étapes sont les étapes de déploiement spécifiques à l'application (copies de fichiers, création de variables d'environnement, …). Ces étapes sont éventuellement associées à une étape de retour arrière (*Undo Step*). Ainsi, en cas d'erreur lors d'une étape d'installation, les étapes de retour arrière sont réalisées afin de restaurer le site client dans son état initial (état cohérent).

Enfin, la dernière étape (*Terminate*) permet de nettoyer les répertoires temporaires du site client utilisés lors des activités précédentes.

2.2. Définition des modèles

Pour représenter l'état des machines impliquées dans la fédération, nous utilisons un fichier XML [Xml] représentant le modèle de site et disponible sur chaque machine cliente. Dans ce fichier est indiqué le type de la machine : serveur d'applications ou site client. Pour les serveurs d'applications, la liste des packages disponibles est mentionnée. Pour les sites clients, l'ensemble des applications déployées et leurs informations (nom, version, URL et dépendances) est donné. Dans cette première version, les propriétés ne sont pas utilisées pour le déploiement, mais des chemins de déploiement sont disponibles. Dans cette implémentation, un seul serveur de déploiement est utilisé (il s'agit du serveur de la fédération) : aucun modèle n'est donc défini pour ce type de machine.

Pour le concept d'application, nous utilisons un modèle d'application, stocké au format d'un fichier XML. Ce descripteur est aussi limité à sa plus simple forme. Il contient le nom et la version de l'application, ainsi que l'URL à laquelle elle est disponible. Il contient aussi la liste de ses dépendances (nom d'application). Nous pouvons noter que ces modèles (application et

site) sont assez basiques. Cette expérimentation, nous a permis d'affiner notre méta-modèle de déploiement, défini précédemment.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://castor.exolab.org/"
      version="0.9.4">
   <xsd:complexType name="Param">
      <xsd:sequence>
         <xsd:element name="value" type="xsd:string"/>
         <xsd:element name="type" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="calculated" type="xsd:boolean" />
   </xsd:complexType>
   <xsd:complexType name="Property">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:string"/>
         <xsd:element name="value" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="calculated" type="xsd:boolean" />
   </xsd:complexType>
   <xsd:complexType name="RoleType">
      <xsd:sequence>
         <xsd:element name="roleName" type="xsd:string"/>
         <xsd:element name="interfaceName" type="xsd:string"/>
         <xsd:element name="methodName" type="xsd:string"/>
         <xsd:element name="param" type="Param"
               minOccurs="0"  maxOccurs="unbounded"/>
         <xsd:element name="comment" type="xsd:string"
               minOccurs="0"  maxOccurs="1"/>
         <xsd:element name="mode" type="xsd:string"
                minOccurs="0"  maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Step">
      <xsd:sequence>
         <xsd:element name="do" type="RoleType" />
         <xsd:element name="verification" type="RoleType"
               minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="undo" type="RoleType"
               minOccurs="0" maxOccurs="1" />
         <xsd:element name="comment" type="xsd:string"
               minOccurs="0"  maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="stepName" type="xsd:string" />
   </xsd:complexType>
<xsd:element name="Install">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="propertiesFile" type="xsd:string"
                  minOccurs="0" maxOccurs="1" />
            <xsd:element name="property" type="Property"
                  minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="extract" type="Step"/>
            <xsd:element name="installStep" type="Step"
                  minOccurs="1" maxOccurs="unbounded" />
            <xsd:element name="terminate" type="Step" />
            <xsd:element name="comment" type="xsd:string"
                  minOccurs="0"  maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure 7.3 Schéma d'un descripteur de déploiement

Un descripteur de déploiement, disponible dans le package de l'application, énonce la liste des étapes ordonnées à réaliser pour déployer l'application. Il s'agit des étapes spécifiques au déploiement de l'application : copie de fichiers, création de variables d'environnement, exécution de fichiers de commandes, etc. Ce descripteur est un fichier XML qui répond à la structure du schéma de la Figure 7.3.

Dans ce descripteur, nous retrouvons les activités du procédé de déploiement que nous avons défini précédemment. Une installation (*Install*) est ainsi composée d'une étape d'extraction (*extract*), d'un ensemble d'étapes d'installation (*installStep*) et d'une étape de terminaison (*terminate*). Elle est aussi associée à un ensemble de propriété (*property*), définies dans un fichier de propriétés (*propertiesFile*), et qui seront utilisées au cours de l'installation. Ces propriétés peuvent être des valeurs fixes ou des valeurs à calculer au moment de l'installation, en fonction du modèle de site.

Une étape (*Step*) a un nom (*stepName*) et est associée à un commentaire (*comment*). Le commentaire est informel et sert simplement de note pour l'utilisateur. Une telle étape est aussi composée de trois parties :

- une réalisation (*do*) permet d'énoncer l'activité à réaliser,
- une vérification (*verification*) facultative permet de vérifier si la réalisation précédente s'est bien déroulée,
- un retour arrière (*undo*) facultatif permet de revenir en arrière si la vérification révèle que la réalisation a causé une erreur.

Chacune de ces parties est définie par un rôle (*RoleType*). Celui-ci permet de déterminer quelle méthode (avec ses paramètres) de quel rôle de la fédération utiliser pour réaliser l'étape. Par exemple, l'utilisateur souhaite faire appel au rôle de gestion de fichiers pour créer un fichier dont il indiquera le nom. Ainsi, pour réaliser les différentes étapes, il a fallu définir un ensemble de rôle et d'outils dans la fédération.

2.3. Fédération pour le déploiement

Les données que nous venons de définir sont utilisées dans une fédération d'outils pour le déploiement [EL01, EVC01-c]. Cette fédération constitue notre environnement de déploiement ORYA. L'environnement est implémenté en java [Java]. Nous commençons par définir l'univers commun avec les concepts d'application, de site client, de serveur de déploiement, de serveur d'applications. Ces concepts utilisent les modèles que nous venons de définir. Il faut ensuite définir les rôles et les outils qui sont nécessaires.

2.3.1. Définition des rôles

Grâce au cahier des charges [MKL+02] défini en collaboration avec un partenaire industriel [Actoll], nous avons identifié et défini un ensemble de rôles nécessaires au déploiement d'une application. Ainsi, parmi ces rôles, nous retrouvons :

- Le rôle de **transfert** permet de réaliser le transfert d'un fichier d'une machine à une autre.
- Le **gestionnaire de registres** permet de créer, modifier ou supprimer des variables d'environnement.

- Le **vérificateur de registres** vérifie l'existence ou la valeur d'une variable d'environnement.
- Le **gestionnaire de fichiers** permet de créer, copier ou supprimer des fichiers.
- Le **vérificateur de fichiers** permet de vérifier l'existence d'un fichier.
- Le **gestionnaire d'exécution** permet d'exécuter un fichier de commandes.
- Le **moniteur de fichiers** de log permet d'accéder à un fichier log et de valider ou non (manuellement) l'étape en cours, en fonction de son contenu.

Nous avons ensuite réalisé les outils implémentant ces rôles. Des aspects permettent de synchroniser les données contenues dans les modèles (fichiers XML) et celles de l'univers commun. Nous avons aussi réalisé d'autres outils. Ces outils complémentaires ne sont pas utilisés au cours du déploiement, mais facilitent le travail amont de l'utilisateur.

2.3.2. Des outils annexes

Des outils ont été implémentés pour faciliter le travail de l'utilisateur. Le premier constitue l'interface de déploiement : il permet de visualiser quelles machines sont connectées, quelles applications sont disponibles et de lancer l'installation d'une application sur une machine cliente.

De plus, comme le montre la Figure 7.4, un éditeur de déploiement permet d'assister l'utilisateur (producteur) dans la définition du descripteur de déploiement spécifique à l'application qui va être disponible.

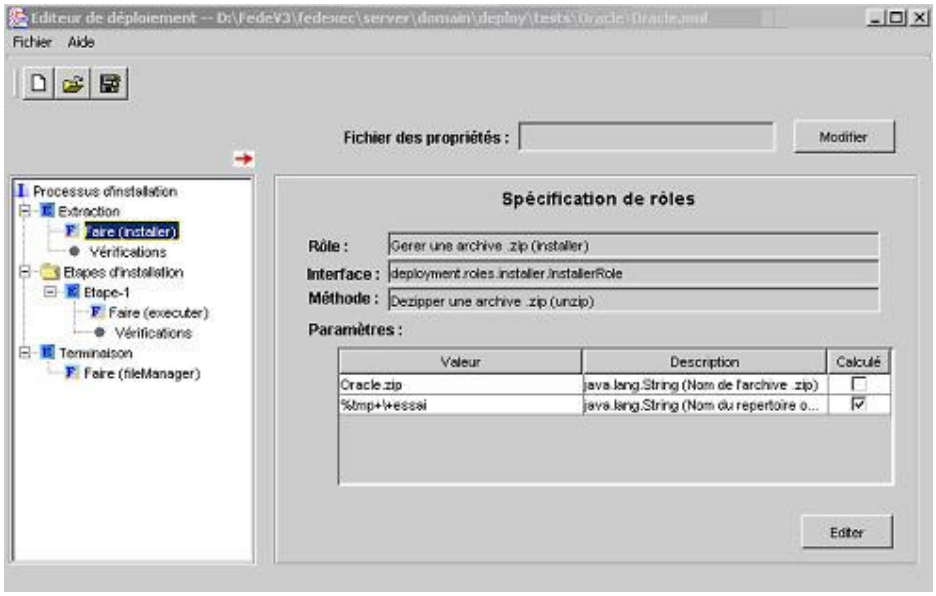

Figure 7.4 Construction du descripteur de déploiement

Une fois l'ensemble de l'information disponible, un outil (Figure 7.5) assiste le producteur dans la construction du package à fournir aux clients.

Figure 7.5 Construction et vérification d'un package

Cet outil permet de rassembler l'ensemble de l'information qui doit être dans le package. De plus, lors de la construction du package, la présence des fichiers nécessaires (en fonction des outils de déploiement utilisés dans le descripteur de déploiement) est vérifiée automatiquement. L'outil informe aussi l'utilisateur sur les propriétés et les outils que devra avoir un site client sur lequel l'application sera déployée.

Enfin, un outil déjà existant peut aussi être utilisé pour suivre l'exécution du procédé de déploiement. Il s'agit de l'agenda d'APEL [DEA98] qui est fourni avec le domaine de procédé, utilisé dans notre environnement de déploiement. Il permet de visualiser la progression du procédé de déploiement et, éventuellement, de valider ou invalider certaines activités manuellement.

L'implémentation de l'ensemble de ces outils et notre environnement de déploiement nous a conduit à mener une expérimentation avec un partenaire industriel.

3. UNE EXPERIMENTATION INDUSTRIELLE DE ORYA - VERSION 1

Afin d'évaluer l'applicabilité de notre environnement de déploiement, nous avons réalisé une expérimentation avec l'entreprise Actoll. Dans cette partie, nous présentons tout d'abord le cas d'étude, puis l'utilisation d'ORYA dans ce contexte et son évaluation.

3.1. Présentation du cas d'étude

Pour évaluer notre approche, nous avons réalisé une expérimentation de notre prototype en milieu industriel. L'entreprise Actoll travaille sur une plate-forme du domaine des transports [Centr], pour gérer les péages autoroutiers et réseaux de transports en commun.

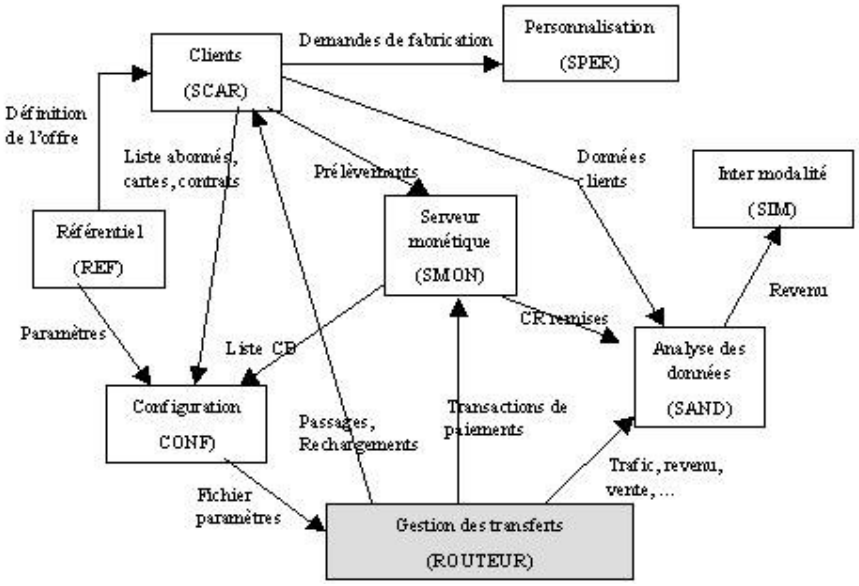

Figure 7.6 Architecture de la plate-forme Centr'Actoll

Cette plate-forme (Centr'Actoll) est composée de différents composants (Figure 7.6). Chacun d'eux a une tâche spécifique :

- **Serveur référentiel** (SREF) : hébergement des fichiers et des données de configuration de l'application (gamme tarifaire, topologie du réseau, habilitation et droits d'accès, …) ;
- **Serveur de configuration** (SCONF) : liaison entre les fichiers de paramètres hébergés dans SREF et les équipements. Il fournit à chaque équipement le fichier de configuration qui lui permet de fonctionner ;
- **Serveur pour la gestion clientèle** (SCAR) : gestion des clients (stocke les clients abonnés), gestion des cartes des clients (pour ceux qui sont abonnés), gestion des contrats à post paiement (prélèvement) ;
- **Serveur pour la personnalisation** (SPER) : personnalisation ou pré-personnalisation des titres de transports (cartes, tickets) à haut débit, utilisé pour délivrer un grand nombre de titres par exemple au démarrage du système ;
- **Serveur monétique** (SMON) : gestion de la monnaie électronique, se spécialisant en :
  - SMON-B : cas des paiements par carte bancaire, cas des paiements par prélèvements ;

  - SMON-P : cas des paiements par cartes privatives (comme *Shell*, *Esso*, *American Express*, *DKV*) ;
  - SMON-PME : cas des paiements par porte monnaie électronique ;
- **Serveur d'analyse des données** (SAND) : analyse des données (statistiques, contrôle des recettes, mise en forme des données pour interprétation avec un ERP comme SAP) ;
- **Serveur d'inter modalité** (SIM) : gestion des réseaux multi / inter-modaux (1 seul ticket pour parking, bus, …) et des compensations (une partie de l'argent encaissé revient à quelqu'un d'autre).

Chacun des composants de la plate-forme s'exécute sur un serveur en utilisant d'autres applications. Par exemple, plusieurs composants utilisent une base de données Oracle. ORYA est utilisé pour installer ces composants sur un ensemble de machines clientes. Les procédés d'installation des composants utilisent les mêmes techniques (exécution de scripts, copie de fichiers, création de variables d'environnement, …). Pour notre expérimentation, nous nous focalisons donc sur les composants SCAR et SMON-B.

3.2. Utilisation d'ORYA

Plusieurs étapes permettent de réaliser un déploiement [Mer03]. La première étape consiste à définir le descripteur de déploiement de l'application et à construire le package correspondant, avec les outils qui sont disponibles pour assister l'utilisateur. Ensuite, le producteur ajoute le package parmi ceux fournis par les serveurs d'applications d'ORYA.

Depuis le serveur de déploiement, l'administrateur peut choisir, à l'aide de l'interface d'ORYA, de déployer une application spécifique sur un site donné. Après ce choix, la fédération et le moteur de procédés [Vil02] pilotent le déploiement en suivant les étapes du procédé de déploiement (définies par l'utilisateur, dans le descripteur de déploiement) [Les03]. Le noyau de la fédération lance chaque outil quand cela est nécessaire et l'utilisateur peut contrôler le déploiement sur la fenêtre de traces associée au procédé.

Sur la demande d'Actoll qui souhaitait contrôler certaines étapes du déploiement manuellement, nous avons défini un mode de déploiement. L'administrateur peut alors choisir trois modes de déploiement différents : manuel, automatique ou semi-automatique. Le mode automatique ne requiert aucune intervention de l'utilisateur. Lors d'un déploiement en mode manuel, l'utilisateur doit valider/invalider manuellement chaque étape du déploiement. Enfin, en mode semi-automatique, certaines activités seront réalisées automatiquement, d'autres demanderont à l'utilisateur d'être validées/invalidées.

Ce travail a été réalisé et testé pour les composants SCAR et SMON-B de la plate-forme Centr'Actoll. Il nous a permis d'évaluer l'applicabilité de notre environnement de déploiement en milieu industriel. De plus, cette expérimentation nous a aussi permis d'identifier les points forts et les points faibles de notre approche.

3.3. Evaluation de l'expérimentation

L'entreprise Actoll a particulièrement apprécié notre collaboration dans le domaine du déploiement. En effet, avant d'utiliser notre environnement, chaque déploiement était réalisé manuellement, notamment en exécutant des fichiers de commandes ou des copies de fichiers.

L'utilisation d'ORYA leur a permis de définir un procédé de déploiement pour différents composants. Ils ont apprécié le fait que ce procédé soit reproductible sur un ensemble de machines cibles sans être modifié.

Ainsi, notre approche démontre que la technologie des procédés basée sur les fédérations fournit un excellent support pour la gestion du déploiement. De plus, elle montre comment un système de déploiement peut être défini avec des techniques spécifiques aux procédés pour répondre aux nouveaux besoins des entreprises dans ce domaine. Elle démontre aussi que l'architecture du système peut être étendue en ajoutant de nouveaux outils au noyau de la fédération : une fois ajoutés, ces outils inter-opèrent.

Cependant, notre expérimentation a aussi exprimé la faiblesse de la modélisation de notre environnement. En effet, il manque une modélisation de l'architecture de l'entreprise pour permettre de déployer une application sur un ensemble de machines (comme les machines du service de comptabilité, par exemple).

De plus, les modèles d'application et de site qui ont été utilisés dans la première version sont relativement pauvres et doivent être étendus. Par exemple, les contraintes imposées par les applications à déployer n'ont pas été utilisées et le choix de l'application à déployer est entièrement réalisé par l'administrateur, en fonction de ses propres connaissances des machines cibles.

En outre, il manque un niveau de politiques de déploiement. En effet, l'administrateur de l'entreprise ne peut pas appliquer de stratégies de déploiement, avec cette première version de notre environnement. En utilisant l'architecture de l'entreprise et les stratégies de déploiement définies par l'administrateur, l'environnement de déploiement doit alors être capable de calculer automatiquement un plan de déploiement pour un ensemble de machines.

Dans la deuxième version d'ORYA, il est donc important de prendre en compte ces points faibles pour les améliorer. La deuxième version d'ORYA est basée sur une approche générique du déploiement [Mer04], pour permettre l'utilisation d'ORYA dans un maximum de contextes.

4. UN ENVIRONNEMENT DE DEPLOIEMENT ORIENTE STRATEGIES : ORYA – VERSION 2

La deuxième version de notre environnement de déploiement est basée sur les concepts de la première. Il est ainsi possible de déployer des applications de manière automatisée. Mais cette deuxième version ne permet plus seulement de déployer une application sur une machine (première version), mais un ensemble d'applications sur un ensemble de machines. Il est alors nécessaire d'introduire les concepts d'entreprise et de stratégie.

4.1. Les bases de ORYA - version 2

Le fonctionnement général de la version 2 reste le même que celui de la version 1, avec l'utilisation des technologies de fédération et de procédé. Les modèles de site et d'application ont cependant été affinés.

```
<?xml version="1.0" encoding="UTF-8"?>
<MachineDescriptor xmlns="http://castor.exolab.org/">
      <name>POUPON</name>
      <machineFunction>ClientSite</machineFunction>
      <userName>merle</userName>
      <property>
            <propertyName>OS</propertyName>
            <propertyValue>winXP</propertyValue>
      </property>
      < property >
            <propertyName>Disk</propertyName>
            <propertyValue>123</propertyValue>
      </property>
      < property >
            <propertyName>Mem</propertyName>
            <propertyValue>512</propertyValue>
      </property>
      <deployedUnit>
            <unit>
                  <name>X</name>
                  <version>4</version>
            </unit>
            <descriptorURL>D:\data\deployed\X\X-4.xml</descriptorURL>
      </deployedUnit>
</MachineDescriptor>
```

Figure 7.7 Exemple de descripteur de machine

Dans cette version du descripteur de machine (Figure 7.7), toutes les informations contenues dans notre méta-modèle de déploiement sont disponibles. Ainsi, une machine est caractérisée par son nom (*POUPON*), son type (*ClientSite*) et la liste de ses utilisateurs (ici, *merle*). De plus, elle est définie par un ensemble de propriétés (*property*), qui sont des couples <nom, valeur>. Dans l'exemple, les propriétés décrivent le système d'exploitation (*OS = winXP*), l'espace disque (*Disk = 123*) et la capacité mémoire (*Mem = 512*). De plus, ce descripteur contient aussi la liste des unités (versions d'applications) qui sont déjà déployées : ici, l'application X - version 4 est déployée.

En ce qui concerne le descripteur d'application (unité de déploiement), il contient aussi toutes les informations nécessaires au déploiement. La Figure 7.8 donne un exemple de la version *1* de l'unité *U*. Cette unité est décrite par trois propriétés (*property*) qui décrivent son implémentation, son type et son interface. De plus, elle impose trois contraintes d'installation, qui traitent du système d'exploitation, de la capacité mémoire et de l'espace disque. Enfin, elle est associée à un plan de déploiement, dont l'URL est indiquée, qui permet son installation.

```
<?xml version="1.0" encoding="UTF-8"?>
<UnitDescriptor xmlns="http://castor.exolab.org/">
      <unitName>U</unitName>
      <unitVersion>1</unitVersion>
      <sourceURL>D:\data\packages\U</sourceURL>
      <property>
            <propertyName>implem</propertyName>
            <propertyValue>java</propertyValue>
      </property>
      <property>
            <propertyName>type</propertyName>
            <propertyValue>devTool</propertyValue>
      </property>
      <property>
            <propertyName>interface</propertynName>
            <propertyValue>txt</propertynValue>
      </property>
      <constraint>
            <activity>INSTALL</activity>
            <cExpr>'OS' E {winXP, win2000}</cExpr>
      </constraint>
      <constraint>
            <activity> INSTALL </ activity >
            <cExpr>'Mem' >= 256</ cExpr >
      </constraint>
      <constraint>
            <activity> INSTALL </ activity >
            <cExpr>'Disk' >= 104</ cExpr >
            <action>ConstraintActionner.substract('Disk', 104)</action>
      </constraint>
      <deployPlan>
            <planName>INSTALL</planName>
            <descriptorURL>D:\data\packages\DocUnit_1-1\InstallPlan.xml</descriptorURL>
      </deployPlan>
</UnitDescriptor>
```

Figure 7.8 Exemple de descripteur d'unité

Après avoir affinés les modèles de site et d'application, il reste à ajouter les concepts liés à l'entreprise : le modèle d'entreprise donne toute l'information nécessaire.

4.2. Modèle d'entreprise

Comme pour les modèles de site et d'application, un fichier XML décrit l'ensemble de l'information liée à l'entreprise. La Figure 7.9 donne un exemple de modèle d'entreprise. L'entreprise G est composée de deux groupes *G1* et *G2*. La structure d'entreprise (sous-groupes et machines) correspond à celle mentionnée dans l'exemple du chapitre précédent (Figure 6.3). De plus, les stratégies de sélection sont liées aux groupes concernés : une stratégie *VERSION-DROITS* pour *G* et une stratégie *VERSION-INTRAGROUPE* pour *G2*.

L'information contenue dans ces descripteurs (machine, unité et entreprise) est synchronisée avec les éléments de l'univers commun et permet ainsi d'effectuer les opérations nécessaires pour préparer le déploiement. Ces informations sont aussi utilisées pour créer les structures d'application et d'entreprise lorsqu'un déploiement est demandé.

```
<?xml version="1.0" encoding="UTF-8"?>
<EnterpriseDescriptor xmlns="http://castor.exolab.org/">
   <enterpriseName>G</enterpriseName>
      <group>
         <groupName>G1</groupName>
         <machine>
            <machineName>M1</machineName>
            <machineFunction>ClientSite</machineFunction>
         </machine>
         <machine>
            <machineName>M2</machineName>
            <machineFunction>ClientSite</machineFunction>
         </machine>
         <strategy>
            <sActivity>INSTALL</sActivity>
            <constraintU>'interface'=multimedia</constraintU>
            <constraintE>JAMAIS</constraintE>
            <strategyType>VERSION-DROITS</strategyType>
         </strategy>
      </group>
      <group>
         <groupName>G2</groupName>
         <machine>
            <machineName>M3</machineName>
            <machineFunction>ClientSite</machineFunction>
         </machine>
         <subGroup>
            <groupName>G3</groupName>
            <machine>
               <machineName>M4</machineName>
               <machineFunction>ClientSite</machineFunction>
            </machine>
            <machine>
               <machineName>M5</machineName>
               <machineFunction>ClientSite</machineFunction>
            </machine>
         </subGroup>
      </group>
      <strategy>
         <sActivity>INSTALL</sActivity>
         <constraintU>'implémentation'=java</constraintU>
         <constraintE>UNITE</constraintE>
         <strategyType>VERSION-INTRAGROUPE</strategyType>
      </strategy>
   </group>
</EnterpriseDescriptor>
```

Figure 7.9 Exemple de descripteur d'entreprise

4.3. Stratégies

En ce qui concerne les stratégies, nous n'avons pas implémenté totalement le langage de description par manque de temps. Nous avons cependant réalisé des exemples de traitement de stratégies. Nous nous sommes intéressés aux stratégies appliquées « statiquement » : nous n'avons donc pas pris en compte les stratégies d'exécution, mais seulement celles de sélection et d'ordonnancement. Ces deux types de stratégies sont traitées en deux étapes indépendantes : sélection complète des unités à déployer, puis ordonnancement total du plan de déploiement.

Dans le cas où les stratégies ne permettent pas une sélection complète (c'est-à-dire une unité et ses dépendances par machine cible), c'est la première unité de la liste des possibilités qui

est choisie. De la même manière, si les stratégies à appliquer ne permettent pas l'ordonnancement total du plan de déploiement, nous considérons que le déploiement doit s'effectuer de manière ordonnée, selon l'ordre de la liste des déploiements unitaires à réaliser. Ces hypothèses permettent de limiter les degrés de liberté du plan de déploiement et n'imposent pas l'utilisation de stratégies d'exécution.

De plus, seules les dépendances obligatoires sont prises en compte dans cette implémentation. Les dépendances sont donc imposées selon les unités qui sont sélectionnées. Les stratégies appliquées aux dépendances sont les mêmes que celles appliquées aux unités.

Nous avons implémenté quelques uns des types de stratégies que nous avons identifiés précédemment (cf. paragraphe 4 du chapitre 6). Parmi les stratégies de sélection, nous avons notamment implémenté :

- les stratégies *VERSION-DROITS*, avec les mots-clés *ALWAYS* (imposer un type d'unité) et *NEVER* (interdire un type d'unité), et,
- les stratégies *VERSION-INTRAGROUPE*, avec le mot-clé *UNIT*, qui impose la même unité de déploiement (version d'une application) pour toutes les machines d'un groupe.

Après l'étape de sélection (complète), un ensemble de plans unitaires (couple <*unité, machine*>) est connu et doit être ordonné à l'aide des stratégies d'ordonnancement. Parmi les stratégies d'ordonnancement, nous avons implémenté :

- les stratégies *CHRONOLOGIE-MACHINES*, avec les mots-clés *ORDERED* (ordonner le déploiement par rapport aux machines cibles) et *AND* (pas d'ordre précis de déploiement), et,
- les stratégies *CHRONOLOGIE-UNITES*, utilisées simplement pour gérer l'ordre de déploiement des dépendances et de l'unité, avec les mots-clés *MANDATORY-DEPENDANCIES* (dépendances obligatoires) et *UNIT* (unité).

Les stratégies d'ordonnancements sont tout d'abord utilisées pour déterminer si les dépendances (*MANDATORY-DEPENDANCIES*) doivent être déployées avant ou après l'unité (*UNIT*). Seuls les déploiements séquentiels sont considérés, c'est pourquoi le mot-clé *PARALLEL* n'intervient pas ici.

4.4. Plan de déploiement

Les stratégies que nous venons de citer sont utilisées pour créer le plan de déploiement global, permettant de déployer une application (*n* unités, chacune étant une version) sur *p* machines.

4.4.1. Structure d'entreprise et structure d'application

Pour simplifier l'implémentation, les structures d'entreprise et d'application sont définies dans le même fichier XML. Un schéma XSD permet de définir quelles informations doit contenir chaque nœud de la structure d'entreprise. La structure d'application fait partie de ces informations, c'est pourquoi elle se trouve dans le même fichier.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://castor.exolab.org/" version="0.9.4">
   <xsd:complexType name="DeployEnterpriseStructDescriptor">
      <xsd:sequence>
         <xsd:element name="deployActivity" type="DeployActivityType"/>
         <xsd:element name="cibleGroupName" type="Node"/>
         <xsd:element name="unitNameToDeploy" type ="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Node">
      <xsd:sequence>
         <xsd:element name="cibleName" type="xsd:string"/>
         <xsd:element name="cibleType" type="CibleType"/>
         <xsd:element name="possibleUnits" type ="DeployUnitsStruct"/>
         <xsd:element name="localStrategy" type="StrategyType" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="heritedStrategy" type="StrategyType" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="son" type="Node" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="property" type="PropertyType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="StrategyType">
      <xsd:sequence>
         <xsd:element name="sActivity" type="DeployActivityType"/>
         <xsd:element name="constraintU" type="xsd:string"/>
         <xsd:element name="constraintE" type="xsd:string"/>
         <xsd:element name="strategyType" type="StrategyCategoryType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="DeployActivityType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="INSTALL"/>
         <xsd:enumeration value="UPDATE"/>
         <xsd:enumeration value="RECONFIGURE"/>
         <xsd:enumeration value="UNINSTALL"/>
         <xsd:enumeration value="ACTIVATE"/>
         <xsd:enumeration value="DESACTIVATE"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="StrategyCategoryType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="VERSION-INTRAGROUPE"/>
         <xsd:enumeration value="VERSION-DROITS"/>
         <xsd:enumeration value="CHRONOLOGIE-MACHINES"/>
         <xsd:enumeration value="CHRONOLOGIE-UNITES"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="CibleType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="MACHINE"/>
         <xsd:enumeration value="GROUP"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="DeployUnitsStruct">
      <xsd:sequence>
         <xsd:element name="possibleUnit" type="UnitNode" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="UnitNode">
      <xsd:sequence>
         <xsd:element name="unitName" type="xsd:string"/>
         <xsd:element name="unitVersion" type="xsd:string"/>
         <xsd:element name="sourceURL" type="xsd:string"/>
         <xsd:element name="property" type="PropertyType" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="constraint" type="ConstraintType" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="mandatory-dependancy" type="UnitNode" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="deployPlan" type="DeployPlanType" minOccurs="0" maxOccurs="6"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Figure 7.10 Schéma d'un descripteur de structure d'entreprise

La Figure 7.10 donne le schéma de la structure d'entreprise qui est utilisée. Le descripteur d'une structure d'entreprise est composé de :

- Une **activité de déploiement** (*deployActivity*) indique quelle activité doit être effectuée (installation, mise à jour, …).
- Un **groupe de machines cibles** (*cibleGroupName*) indique sur quelle(s) machine(s) le déploiement doit être réalisé.
- Des **noms d'applications** (*unitNameToDeploy*) indiquent quelles applications déployer. Comme nous nous sommes particulièrement intéressés au cas du déploiement d'une application sur un groupe de machines, une seule application (*unitNameToDeploy*) sera utilisée.

La cible de déploiement est représentée par un *Nœud*. Un *Nœud* est identifié par son nom et peut être un groupe ou une machine de l'entreprise. Chaque *Nœud* est associé à une structure d'application (*unitesPossibles*) et un ensemble de stratégies locales (*localStrategy*) et/ou héritées (*heritedStrategy*).

Le structure d'application (type *DeployUnitsStruct*) est composée d'un ensemble d'unités possibles. Chaque unité (type *UnitNode*) est identifiée par son nom et sa version. Elle est décrite par des propriétés (*property*), impose des contraintes (*constraint*) et a des dépendances obligatoires (*mandatory-dependancy*).

Le descripteur de la structure d'entreprise est créé au moment où un déploiement est demandé par l'administrateur. Il permet de rassembler l'information à traiter pour pouvoir réalisé le déploiement. Ainsi, cette structure évolue au cours de l'application des diverses stratégies. A la fin de l'étape de sélection, chaque nœud est associé à la structure d'application qui contient les unités qu'il va recevoir.

4.4.2. Sélection des unités

Lorsqu'un utilisateur demande le déploiement d'une application (ensemble de versions) sur un groupe de machines de l'entreprise, le descripteur de la structure d'entreprise est créé en fonction du groupe cible et des versions de l'application (unités de déploiement) qui sont disponibles. L'ensemble des unités disponibles forme la structure d'application qui est associée à la racine de la structure d'entreprise. Il faut ensuite sélectionner quelle(s) unité(s) déployer sur chaque machine (cible finale de déploiement) du groupe.

L'étape de sélection utilise alors les stratégies de sélection de chaque nœud en appliquant la méthode suivante.

```
private DeployUnitsStruct select (Node noeud, String deployActivity);
```

Nous avons vu dans le paragraphe 5.1.1 du chapitre 6 le principe de cet algorithme :

- vérification des contraintes des unités par rapport aux propriétés des machines cibles et
- application des stratégies de groupes et de machines.

Dans cette partie, nous présentons plus particulièrement la manière de traiter les stratégies.

Ainsi les stratégies de type *VERSION-DROITS* sont traitées par la méthode *traiterStrategiesDroits*. Selon, le mot-clé utilisé (*NEVER* ou *ALWAYS*), cette méthode consiste à supprimer des unités possibles en regardant les propriétés de chacune. Les

méthodes *removePossibleUnitWithProperty* et *keepOnlyUnitsWithProperty* permettent alors de supprimer ou de ne conserver que les unités remplissant la contrainte-Unités (*constraintU*) de la stratégie.

```
private DeployUnitsStruct traiterStrategiesDroits (
                        DeployUnitsStruct struct, Vector strategies) {
   UnitNode[] unitesPossibles = struct.getPossibleUnit();
   int j = 0; int k;
   while (j < strategies.size()) {
      if ( ((StrategyType) strategies.get(j)).getConstraintE().
                                          equalsIgnoreCase("NEVER") ){
         struct = removePossibleUnitWithProperty(struct,
                 ((StrategyType)strategies.get(j)).getConstraintU() );
      }
      else if ( ((StrategyType) strategies.get(j)).
                     getConstraintE().equalsIgnoreCase("ALWAYS") ){
         struct = keepOnlyUnitsWithProperty(struct,
                 ((StrategyType)strategies.get(j)).getConstraintU() );
      }
      j++;
   }
   return struct;
}
```

Les stratégies *VERSION-INTRAGROUPE* ont aussi été traitées, pour permettre le déploiement de la même unité sur un groupe de machines (mot-clé *UNIT*). La méthode *traiterStrategiesMemeVersion* est appelée quand la sélection des fils d'un nœud est terminée. Si la même version doit être déployée, seules les unités étant possibles pour tous les fils sont conservées. Sinon (pas de stratégie *VERSION-INTRAGROUPE*), la structure d'application (unités possibles) du nœud est élaguée en ne conservant que les unités qui sont possibles pour au moins l'un des fils.

```
private DeployUnitsStruct traiterStrategiesMemeVersion(
                                    Node noeud, boolean memeVersion) {
   int i;
   UnitNode[] upossTab= noeud.getPossibleUnits().getPossibleUnit();
   Vector toKeep = new Vector();
   if ( memeVersion ){
      // pour chaque unité possible du nœud:
      i = 0;
      while ( i < upossTab.length ) {
         // ne conserver que les unités possible pour tous les fils
         if ( possiblePourTous (upossTab[i], noeud.getFils()) ) {
```

```
            toKeep.add(upossTab[i]);
         }
         i++;
      }
   }
   else {    // sinon (pas de meme version):
      // supprimer les unites qui ne sont possibles pour aucun fils
      i = 0;
      while ( i < upossTab.length ) {
         if ( possiblePourAuMoinsUn (upossTab[i], noeud.getFils()) ) {
            toKeep.add(upossTab[i]);
         }
         i++;
      }
   }
   // retourner la structure correspondant au vector toKeep
   DeployUnitsStruct toReturn = new DeployUnitsStruct();
   UnitNode[] toKeepTab = new UnitNode[toKeep.size()];
   i = 0;
   while ( i < toKeep.size() ) {
      toReturn.setUnitePossible(i, (UnitNode) toKeep.get(i));
      i++;
   }
   return toReturn;
}
```

Après l'application des stratégies de sélection, si plusieurs unités restent possibles pour un nœud, c'est la première de la liste qui est choisie. Après la sélection, le fichier XML reflétant la structure d'entreprise contient les unités à déployer pour chaque machine cible. Il reste ensuite à construire la structure de plan en ordonnant des différents déploiements unitaires.

4.4.3. Ordonnancement des plans de déploiement unitaires

A partir de la structure d'entreprise, la structure de plan peut être construite. La structure de plan est stockée dans un descripteur XML (Figure 7.11). Ce descripteur permet de savoir à quel déploiement il correspond, avec l'activité de déploiement (*deployActivity*), le nom du groupe cible (*cibleName*) et le nom de l'application à déployer (*unitName*). Les autres informations (*nodePlan*) correspondent à la structure de plan.

Les nœuds sont soit des nœuds avec des fils (*NodePlanType*), soit des feuilles (*LeafType*). Les nœuds avec des fils sont représentés par un terme exprimant un ordre (*orderWord*). Les feuilles indiquent un plan unitaire, avec le nom de la machine (*machineName*), le nom (*unitName*) et la version (*unitVersion*) de l'unité de déploiement, l'URL (*sourceURL*) à laquelle elle est disponible et l'activité de déploiement (deployActivity) qui doit être réalisée.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="http://castor.exolab.org/" version="0.9.4">
   <xsd:complexType name="DeployPlanStructDescriptor">
      <xsd:sequence>
         <xsd:element name="deployActivity" type="DeployActivityType"/>
         <xsd:element name="cibleName" type ="xsd:string"/>
         <xsd:element name="unitName" type ="xsd:string"/>
         <xsd:element name="nodePlan" type ="NodePlanType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="NodePlanType">
      <xsd:sequence>
         <xsd:element name="orderWord" type="OrderWordType"/>
         <xsd:element name="sonNode" type="NodePlanType" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="sonLeaf" type ="LeafType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="LeafType">
      <xsd:sequence>
         <xsd:element name="machineName" type="xsd:string"/>
         <xsd:element name="unitName" type="xsd:string"/>
         <xsd:element name="unitVersion" type="xsd:string"/>
         <xsd:element name="sourceURL" type="xsd:string"/>
         <xsd:element name="deployActivity" type="DeployActivityType"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="OrderWordType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="ORDERED"/>
         <xsd:enumeration value="AND"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="DeployActivityType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="INSTALL"/>
         <xsd:enumeration value="UPDATE"/>
         <xsd:enumeration value="RECONFIGURE"/>
         <xsd:enumeration value="UNINSTALL"/>
         <xsd:enumeration value="ACTIVATE"/>
         <xsd:enumeration value="DESACTIVATE"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

Figure 7.11 Schéma d'un descripteur de structure de plan

La méthode suivante permet de créer un nœud représentant un plan de déploiement.

```
public NodePlanType orderNodePlan (Noeud noeud,String deployActivity);
```

La construction du plan de déploiement dépend du type du nœud de la structure d'entreprise qui est parcourue. Pour les nœuds de type *GROUP*, il faut tout d'abord transmettre les stratégies de type *CHRONOLOGIE-UNITES*. En effet, celles-ci seront utilisées au niveau des machines pour déterminer si le déploiement des dépendances doit précéder ou suivre celui de l'unité. Ensuite, il faut consulter les stratégies de type *CHRONOLOGIE-MACHINES* pour déterminer le type du nœud à créer (*ORDERED*, *AND*, …). Puis il faut effectuer les appels récursifs permettant de construire les plans fils et ajouter ses plans dans l'ordre indiqué par la stratégie.

Au niveau d'un nœud de type *MACHINE*, si l'unité à déployer a des dépendances, il faut consulter les stratégies de type *CHRONOLOGIE-UNITES*, pour déterminer l'ordre de déploiement. En fonction de ces stratégies, il faut créer le nœud correspondant et les feuilles du plan de déploiement. Si l'unité à déployer n'a aucune dépendance, il suffit simplement de créer la feuille. Les feuilles sont créées à partir des informations disponibles dans la structure

d'entreprise. Le code suivant est une partie de la méthode qui permet de créer un nœud à partir d'un nœud de la structure d'entreprise (contenu dans la variable *fNoeud*). Dans cette partie, il s'agit de créer les plans fils d'un nœud dont la racine est un *orderWord*.

```
if (fNoeud.getCibleType().toString().equalsIgnoreCase("GROUP") ) {
   // cas group : appel récursif
   noeudPlanFils = this.orderNodePlan(fNoeud, deployActivity);
   toReturn.addSonNode(i, noeudPlanFils);
}
else if(fNoeud.getCibleType().toString(). equalsIgnoreCase("MACHINE"))
{  StrategyType chronoUStrat =
                this.getStrategieChronoU(fNoeud, deployActivity);
   if (fNoeud.getUnitesPossibles().getUnitePossible()[0].
                                          getDependancyCount() != 0 ) {
      /* cas machine, avec dépendances => plusieurs unités =>
      nœud NodePlanType=> appel à la méthode traitant les déploiements
      sur une machine et ajout du fils au plan a retourner */
      noeudPlanFils = createNodeMachine (
            chronoUStrat.getStrategyChoice(), fNoeud, deployActivity);
      toReturn.addSonNode(i, noeudPlanFils);
   }
   else {
      /* cas machine, sans dépendances => une seule unité =>
      nœud LeafType => création d'une feuille */
      LeafType noeudPlanFeuille = new LeafType();
      noeudPlanFeuille.setDeployActivity(
                     fede.orya.model.deployPlanStructDescriptor.types.
                     DeployActivityType.valueOf(deployActivity));
      noeudPlanFeuille.setMachineName(fNoeud.getCibleName());
      noeudPlanFeuille.setUnitName(
                  fNoeud.getUnitesPossibles().getUnitePossible()[0].
                                                      getUnitName());
     noeudPlanFeuille.setUnitVersion(fNoeud.getUnitesPossibles().
                              getUnitePossible()[0].getUnitVersion());
     noeudPlanFeuille.setSourceURL(fNoeud.getUnitesPossibles().
                                getUnitePossible()[0].getSourceURL());
     toReturn.addSonLeaf(i, noeudPlanFeuille);
   }
  }
```

Une fois le descripteur de plan entièrement construit, il représente le plan de déploiement et sert de base à l'exécution du déploiement demandé.

4.4.4. Exécution du plan de déploiement

Le descripteur de structure de plan construit dans la section précédente sert de support à l'exécution du plan de déploiement.

Dans cette implémentation, nous n'avons utilisé que des nœuds de type *AND* ou *ORDERED*. Pour simplifier l'exécution du plan de déploiement global, le nœud *AND* est interprété comme le nœud *ORDERED* : aucune stratégie d'exécution n'est utilisée pour déterminer un autre ordre.

La structure de plan est donc parcourue en profondeur d'abord et depuis les fils de gauche vers les fils de droite. Dès que le parcours atteint une feuille, le plan unitaire correspondant est exécuté. L'URL de celui-ci renvoie aux instructions nécessaires à son exécution. Les plans unitaires (procédés) sont exécutés de la même manière que dans la version 1 d'ORYA.

L'exécution du plan de déploiement global reste assez simple dans cette version d'ORYA, avec un parcours de la structure et l'exécution des procédés de déploiement unitaires. Dans une prochaine version, la structure de plan pourra être utilisée pour construire un procédé représentant le plan de déploiement global. De plus, un langage de procédé [Les03] pourra être intégré pour définir les plans de déploiement unitaires.

4.5. Outils

Des outils permettent d'exécuter les différentes étapes de chaque déploiement unitaire (cf. paragraphe 2.3.1), mais d'autres outils permettent aussi de gérer le déploiement et les différents modèles de description de données.

Ainsi, un outil est lancé au démarrage de la fédération de déploiement (ORYA), sur la machine exécutant le moteur de la fédération (serveur de déploiement). Cet outil permet d'avoir la vision des données de l'entreprise (groupes, machines connectées, …) et de les gérer. Il permet aussi de demander le déploiement d'une application sur un groupe de machines. Le nom de l'application est choisi parmi la liste de celles disponibles sur les serveurs d'applications. Le nom du groupe de machines est choisi parmi la liste de ceux qui composent l'entreprise.

De la même manière, un outil est lancé sur chaque site client lorsqu'il se connecte à la fédération. Cet outil permet d'avoir la vision des caractéristiques de la machines et, éventuellement, d'en ajouter de nouvelles (propriétés). Il permet aussi de connaître les unités qui sont déployées.

Les serveurs d'applications, quant à eux utilisent un outil qui permet d'ajouter/supprimer des unités à déployer (différentes versions d'applications). Il est aussi possible de consulter l'information qui les concerne.

5. EVALUATION DE LA VERSION 2

Cette deuxième version d'ORYA permet toujours d'automatiser le déploiement. L'apport essentiel, par rapport à la première version, est la gestion du niveau entreprise et la gestion des stratégies.

Il est maintenant possible de regrouper les différentes machines de l'entreprise en groupes (et sous-groupes) et de déployer une application (disponibles en plusieurs versions) sur un groupe de machines : chaque machine reçoit alors une version qui lui convient.

L'utilisation de stratégies permet de personnaliser le déploiement selon les besoins de l'entreprise. En effet, la sélection des unités à déployer peut être modifiée selon certaines caractéristiques ou préférences de l'entreprise. L'ordonnancement du plan de déploiement peut aussi être différent selon les stratégies appliquées, de même que son exécution.

L'application des stratégies permet de sélectionner les unités à déployer et d'ordonnancer les divers plans de déploiement unitaires (une unité sur une machine cible) pour créer automatiquement le plan de déploiement global, permettant de déployer une applications (*n* unités) sur un groupe de *p* machines cibles

Cette implémentation doit cependant encore évoluer pour permettre la mise en œuvre complète des stratégies de déploiement, et envisager une utilisation industrielle. Nous avons cité notamment la nécessité d'extensions au niveau des procédés (langage de description de procédé), représentant les plans de déploiement.

Côté stratégies, il faut tout d'abord implémenter complètement le langage de description de stratégies (cf. paragraphe 6 du chapitre 6). Ainsi, le langage de stratégies permettra à l'utilisateur de définir les types de stratégies qu'il souhaite, alors que dans l'implémentation actuelle, seuls quels types de stratégies sont traités. De plus, il faut aussi travailler sur la gestion des différentes interactions avec les stratégies (cf. paragraphe 7 du chapitre 6).

Enfin, avant une utilisation industrielle, il est intéressant d'ajouter la possibilités de déployer *m* applications (chacune disponible en *n* versions) sur un groupe de *p* machines. Il est aussi nécessaire de s'intéresser au cas du déploiement d'applications distribuées. En effet, nous avons traité le cas du déploiement d'une application en plusieurs exemplaires, et non le cas du déploiement d'une application distribuée sur plusieurs machines. Dans ce cas, de nouvelles stratégies pourraient être définies.

Enfin, il est aussi possible de définir un nouvel outil qui permettrait de construire la structure d'entreprise à utiliser lors de chaque déploiement. En effet, dans cette implémentation, nous avons systématiquement utilisé la hiérarchie décrite dans le modèle d'entreprise. Un tel outil permettrait de modifier cette structure selon les besoins de chaque déploiement, par exemple, en supprimant ou en rajoutant des machines au groupe cible de déploiement, ou en sélectionnant les stratégies à appliquer.

Le but de cette implémentation n'était pas d'être exhaustive quant aux fonctionnalités offertes, mais de mesurer la faisabilité de la mise en œuvre de stratégies de déploiement. Cette implémentation a ainsi pu permettre de réaliser des tests, avec des types de stratégies spécifiques.

L'utilisation de stratégies de déploiement permet de prendre en compte les différentes préférences de l'entreprise pour gérer la flexibilité du déploiement (choix des unités à déployer, ordonnancement, etc.). La classification des stratégies (en différents types, avec des caractéristiques particulières pour chacun) permet d'automatiser leur traitement, et donc d'automatiser la préparation du déploiement. La construction d'un plan global de déploiement est alors réalisable de manière automatisée.

Malgré son manque de possibilités au niveau des stratégies (langage), cette implémentation d'ORYA montre la faisabilité et l'utilité d'un environnement de déploiement orienté stratégies.

6. SYNTHESE

Dans ce chapitre, nous avons présenté la réalisation de notre environnement de déploiement ORYA. Comme pour l'approche « théorique », l'approche « pratique » a été réalisée en deux étapes : l'automatisation du déploiement, puis la gestion du niveau entreprise avec des stratégies de déploiement. Deux versions ont donc été réalisées.

La première version d'ORYA a permis de mettre en place un environnement pour le déploiement automatisé. Cette version est implémentée comme une fédération utilisant la technologie des procédés. Elle se limite au déploiement d'une application sur une machine. Elle a permis de montrer que les technologies des fédérations et des procédés sont une bonne approche pour le déploiement automatisé d'applications.

La deuxième version d'ORYA intègre le niveau entreprise et les stratégies de déploiement. Elle permet le déploiement d'une application (disponible en n versions) sur un groupe de p machines. En appliquant les stratégies de déploiement, il est possible de sélectionner quelles unités (versions d'applications) déployer, puis de créer (et ordonnancer) un plan de déploiement global. Même si cette deuxième version nécessite encore beaucoup de travail pour une utilisation industrielle, elle montre l'utilité des stratégies pour le déploiement.

CHAPITRE 8 : CONCLUSION ET PERSPECTIVES

Préambule :

Ce chapitre conclut le travail exposé et présente les perspectives pour la suite.

---

1. CONCLUSION

Dans cette partie, après un bref rappel de notre problématique, nous concluons sur notre travail et nos principales contributions.

1.1. Problématique du déploiement à grande échelle

La problématique du déploiement à grande échelle est particulièrement riche et complexe. En effet, plusieurs problèmes peuvent être pris en compte. Le déploiement exige alors la connaissance de divers domaines à différents instants : gestion du cycle de vie du déploiement, hétérogénéité des machines cibles, etc.

Nous nous sommes intéressés à fournir un support pour le déploiement, le plus proche possible des besoins des entreprises. L'un des objectifs est l'automatisation du déploiement. Ainsi, il est possible de gérer un grand nombre de machines.

Pour les entreprises, il est aussi nécessaire de prendre en compte leur organisation interne. En effet, le choix du déploiement peut se porter sur un groupe précis de machines cibles : machines sous *Windows*, machines du service financier, etc. Le déploiement dépend alors de stratégies de déploiement qui sont définies par l'entreprise.

Pour prendre en compte l'organisation de l'entreprise et les stratégies de déploiement qu'elle souhaite appliquer, nous utilisons une architecture à trois niveaux. Un niveau entreprise a été ajouté entre le niveau du producteur/fournisseur et le niveau de la cible. Ainsi, l'entreprise peut effectuer ses propres traitements et personnaliser le déploiement en fonction de ses besoins et/ou préférences.

De plus, les différentes machines cibles d'un groupe peuvent avoir des caractéristiques différentes et exiger des versions différentes d'une application (unités de déploiement différentes). Il faut alors pouvoir gérer le déploiement d'unités différentes sur un groupe de machines.

Les unités de déploiement peuvent aussi avoir les relations de dépendances entre elles. Il faut pouvoir assurer que le déploiement d'une unité n'est effectuée que si celle-ci pourra fonctionner correctement. Ceci implique notamment le déploiement des dépendances si cela est nécessaire. Mais il faut aussi s'assurer que les contraintes imposées par l'unité (espace disque, capacité mémoire, système d'exploitation, …) sont vérifiées par les caractéristiques de la machine cible.

Enfin, nous avons choisi une approche utilisant un plan de déploiement. Ce plan doit pouvoir être construit automatiquement, selon le déploiement à effectuer et les informations disponibles. Il faut alors prendre en compte les préférences de l'entreprise pour la sélection des unités à déployer : même unité sur un groupe de machine, unités implémentées en java, etc. Il faut ensuite étudier les préférences concernant l'ordonnancement du déploiement : déployer sur une machine avant une autre, déployer les dépendances avant l'unité, etc.

L'entreprise peut aussi préférer effectuer différents choix de déploiement le plus tard possible, c'est-à-dire au moment de l'exécution du plan de déploiement. Dans ce cas, le plan de déploiement doit refléter ces divers degrés de libertés. Par exemple, si une unité de

déploiement ne doit être déployée que sur l'une des machines d'un groupe, le choix pourra se porter sur celle dont la capacité mémoire est la plus importante.

Dans la problématique sur le déploiement à grande échelle, nous nous sommes focalisés sur l'automatisation du déploiement à grande échelle pour les entreprises. Pour cela, nous avons basé notre approche sur la modélisation des concepts et la réutilisation. De plus, pour prendre en compte les préférences et/ou contraintes de l'entreprise, nous avons mis en place les stratégies de déploiement.

1.2. Principales contributions et résultats obtenus

Dans ce paragraphe, nous résumons plus particulièrement les contributions que nous avons apportées pour traiter la problématique qui vient d'être exposée.

1.2.1. Modélisation des concepts de déploiement et réutilisation

Nous avons commencé notre travail par la modélisation des concepts liés au déploiement. Pour cela, nous avons défini un méta-modèle de déploiement qui se divise en trois parties :

- une partie est liée aux **unités de déploiement** et à leur description,
- une partie est liée aux **machines cibles** et à la définition de leurs caractéristiques et de leur état (en terme d'unités déjà déployées),
- une partie est liée à l'environnement de **l'entreprise**, à la structuration hiérarchique des différentes cibles de déploiement (machines) et aux stratégies de déploiement à utiliser.

Notre méta-modèle contient donc les concepts principaux du déploiement :

- une **unité de déploiement** est l'entité que l'on cherche à déployer.
- une **machine** est la cible finale de déploiement, qui va recevoir une unité de déploiement.
- un **groupe** rassemble un ensemble de machines qui doivent être traitées ensembles pour le déploiement.

A ces concepts s'ajoutent des concepts complémentaires :

- les **propriétés** décrivent les unités de déploiement et les machines cibles.
- les **contraintes**, associées aux unités de déploiement, indiquent quelles caractéristiques doit avoir une machine cible pour que l'unité puisse être déployée et fonctionner correctement.
- les **stratégies** de déploiement, associées aux groupes et machines, expriment des contraintes (besoins et/ou préférences) définies par l'entreprise et permettent de construire le plan de déploiement.

La relation de dépendance entre unités peut aussi s'ajouter à ces notions. Il s'agit d'exprimer quelle(s) unité(s) sont nécessaires pour le déploiement et le bon fonctionnement d'une unité à déployer.

Notre méta-modèle de déploiement s'abstrait au maximum de tous les concepts spécifiques liés à chacun de ces domaines. L'utilisateur peut ensuite créer ses propres modèles, conformes au méta-modèle. Il peut aussi choisir de réutiliser des modèles déjà définis dans d'autres contextes. Par exemple, pour le déploiement d'applications à base de composants, l'utilisateur peut réutiliser le modèle à composants (OSGi, CCM, …) qui définit ses applications.

Une telle approche dirigée par les modèles permet une bonne réutilisation des concepts ainsi définis. La partie générique est exposée dans notre méta-modèle de déploiement. L'utilisateur a alors à déterminer les parties spécifiques, en définissant les modèles qu'il souhaite utiliser. Il n'a ensuite plus qu'à établir les « correspondances » entre ces données et les concepts du méta-modèle.

La mise en œuvre de notre méta-modèle et son implémentation utilise la technologie des fédérations. Ce choix permet de simplifier la réutilisation. En effet, il est possible de composer des domaines déjà existants entre eux pour former une application. C'est ainsi que nous avons construit notre environnement de déploiement.

1.2.2. Plan de déploiement et automatisation

Dans notre approche, nous avons choisi de construire un plan de déploiement. Ce plan est tout d'abord construit à partir des informations disponibles. Il est ensuite exécuté pour effectuer les différentes étapes du déploiement.

Un plan de déploiement unitaire est fourni par le producteur/fournisseur de l'unité à déployer. Ce plan constitue la brique de base d'un plan de déploiement global. Il indique les instructions à réaliser pour déployer l'unité à laquelle il est associé, sur une machine cible.

Un plan de déploiement global est construit de manière automatisée à partir des informations de l'entreprise (caractéristiques des machines, stratégies de déploiement, …). Un tel plan permet de déployer un ensemble d'unités de déploiement, sur un ensemble de machines cibles.

Le plan de déploiement global doit refléter l'application des stratégies de déploiement, mais aussi les différents degrés de liberté qui sont encore possibles pour son exécution. L'application des stratégies se reflète dans la sélection des unités à déployer et dans l'ordonnancement. L'ordre de déploiement peut être partiel ou total. Un ordre total indique alors un degré de liberté à résoudre au moment de l'exécution du plan.

Dans la structure de plan, les degrés de libertés sont représentés par un type de nœud particulier. Lors de l'exécution du plan global, ces nœuds sont traités différemment. Le choix possible est réalisé en fonction des stratégies d'exécution. Ainsi, par exemple, la sélection d'une unité à déployer peut se porter vers celle qui demande le moins de ressources.

L'utilisation d'un plan de déploiement, qui est construit puis exécuté, ajoute des cas d'erreur possibles. En effet, l'environnement de déploiement peut avoir été modifié entre le moment où le plan est construit et celui où il est exécuté. Par exemple, l'espace disque nécessaire peut ne plus être disponible.

Cependant, l'utilisation d'un plan de déploiement apporte aussi des avantages. Ainsi, tout déploiement débutant son exécution peut arriver à son terme sans erreur (aux erreurs d'évolution près). Par exemple, les cas où les caractéristiques de la (des) machine(s) cible(s)

ne sont compatibles avec aucune des unités de déploiement disponibles, sont éliminés. En effet, dans ce cas, le plan de déploiement global ne peut pas être établi. Si nous prenons le même exemple, dans une configuration où le déploiement est réalisé sans construire de plan, des étapes peuvent avoir été effectuées avant de s'apercevoir de l'incompatibilité entre l'unité et la cible. Dans ce cas, il faudra « défaire » ces étapes pour revenir à un état cohérent de l'environnement de déploiement. Un plan de déploiement est alors intéressant pour éviter de réaliser (et donc de défaire) des opérations inutiles.

Enfin, un plan de déploiement, avec l'utilisation de la technologie des procédés, permet d'automatiser le déploiement. En effet, une fois le plan construit, il ne reste qu'à suivre les instructions qu'il contient pour réaliser le déploiement. Ces étapes peuvent alors être réalisées de manière automatisée, sans l'intervention de l'administrateur. Cependant, celui-ci peut intervenir à des moments clés du déploiement pour valider ou invalider certaines étapes.

1.2.3. Stratégies de déploiement et personnalisation

Durant la phase de déploiement, l'utilisateur doit gérer de la flexibilité à différents niveaux. En effet, plusieurs unités de déploiement peuvent être disponibles, plusieurs ordonnancements peuvent être choisis, etc. Pour permettre de gérer cette flexibilité, nous avons défini des stratégies de déploiement. Les stratégies permettent alors de personnaliser les activités de déploiement, selon les unités à déployer et les contraintes imposées par l'entreprise.

Une stratégie permet d'exprimer une contrainte sur les unités à déployer et une contrainte imposée par l'entreprise sur sa structure. Elle est associée à une activité du cycle de vie du déploiement.

La contrainte sur les unités (*contrainte-Unités*) permet de séparer l'ensemble des unités possibles en deux sous-ensembles. Des traitements différents sont appliqués sur ces deux sous-ensembles en fonction de la contrainte imposée par l'entreprise (*contrainte-Entreprise*).

Nous avons identifié plusieurs catégories et types de stratégies. Chacune des catégories de stratégies intervient à des instants différents de la préparation du déploiement. Ainsi, des stratégies sont utilisées pour :

- sélectionner les unités à déployer parmi celles disponibles,
- ordonnancer les différents plans de déploiement unitaires (une unité sur une machine),
- traiter les derniers degrés de libertés, lors de l'exécution du plan.

L'application et le traitement de ces différentes stratégies permettent donc de construire un plan de déploiement global. Celui-ci permet de déployer un ensemble d'unités sur un ensemble de machines, en respectant les contraintes imposées par l'entreprise.

Pour décrire les stratégies de déploiement, un langage de description de stratégies a été défini. Le langage permet tout d'abord de définir chaque type de stratégie, selon les caractéristiques que nous avons indiquées (position, portée, visibilité, propagation, précédence) et d'établir comment traiter les stratégies de ce type. Le langage permet ensuite d'exprimer chaque stratégie, avec son type, la *contrainte-Unités*, la *contrainte-Entreprise* et l'activité de déploiement à laquelle elle se rapporte.

Enfin, nous avons identifié différentes interactions concernant les stratégies : la composition de stratégies, les conflits entre stratégies et les interactions entre stratégies et dépendances de l'unité à déployer. Ces interactions fournissent des axes de recherche pour les perspectives de travail.

Les stratégies semblent être un bon moyen de personnaliser le déploiement selon les contraintes imposées par l'entreprise. Avec le langage, l'utilisateur peut décrire lui-même les stratégies dont il a besoin. Les stratégies ne sont alors plus imposées par un outil de déploiement, comme c'est souvent le cas dans les outils existants, mais elles sont définies et utilisées selon les besoins de chaque entreprise.

2. PERSPECTIVES

Dans cette partie, nous indiquons quelques perspectives et axes futurs de recherche. Ces travaux sont séparés en deux sections : les perspectives d'évolution de notre environnement ORYA, puis les perspectives liées à l'utilisation de ces travaux dans divers projets de recherche.

2.1. Perspectives d'évolution

Nous avons déjà cité divers points dans les chapitres précédents. Nous présentons ici quelques axes principaux (« théoriques » et/ou « pratiques ») pour les perspectives de travail.

2.1.1. Travail autour des stratégies

Dans le chapitre 6, nous avons identifié plusieurs types d'interactions avec les stratégies. Qu'il s'agisse de composition de stratégies, de conflit entre stratégies ou d'interaction avec les dépendances, le travail mérite d'être approfondi. En effet, aucun mécanisme ne permet, pour le moment, de gérer ces interactions.

Il faut alors étudier la possibilité de définir des méta-stratégies. Une méta-stratégie permettrait de guider les interactions entre les stratégies, et leur application. Ainsi, selon les opérations de déploiement, les stratégies pourraient être utilisées différemment.

Par exemple, une méta-stratégie pourrait définir dans quel ordre de priorité appliquer les différentes stratégies. Une autre pourrait définir quelles stratégies appliquer aux dépendances de l'unité à déployer : celles appliquées à l'unité ou celles spécifiques à chaque dépendance.

Des méta-stratégies pourraient aussi définir comment composer les différents types de stratégies, comme, par exemple, indiquer quelles stratégies de sélection peuvent encore être utilisées lors de l'ordonnancement ou de l'exécution du plan de déploiement.

Dans les travaux à court terme, nous pensons donc étudier l'utilité et la faisabilité de méta-stratégies pour le déploiement. Ce travail comporte non seulement, la partie « théorique » traitant notamment la cohérence lors de composition de stratégies, mais aussi l'implémentation des mécanismes permettant de mettre en œuvre des méta-stratégies. Avec de tels mécanismes, il sera alors possible d'assurer que l'application des stratégies utilisées ne posera aucun problème d'interaction.

2.1.2. Structures de déploiement

Dans nos exemples, nous n'avons utilisé que des parties du modèle d'entreprise, pour construire la structure d'entreprise. Cependant, nous pouvons imaginer que pour certaines opérations de déploiement, la structure d'entreprise à utiliser doit être entièrement reconstruite.

Par exemple, la structure d'entreprise peut être le résultat de la combinaison de caractéristiques logicielles (machines fonctionnant sous *Windows*) et de caractéristiques administratives (machines des chefs de services, des secrétaires, …). Dans ce cas, il faut imaginer les langages, mécanismes et outils qui peuvent permettre une telle construction.

En ce qui concerne la structure de plan, il faut approfondir le cas des nœuds exprimant un choix. En effet, même si ce type de nœuds n'a pas été implémenté, certains choix peuvent encore être possibles lors de l'exécution du plan de déploiement. Dans ce cas, la structure de plan doit refléter les différents degrés de liberté encore disponibles.

Un tel nœud doit alors être traité différemment d'un nœud où aucun choix n'est possible. Il faut connaître sur quelles données (unité, cible de déploiement, …) doit se porter le choix à réaliser : s'agit-il de sélectionner une unité de déploiement pour une machine cible, s'agit-il de sélectionner une seule machine cible parmi un groupe, … D'autres informations peuvent aussi être ajoutées pour permettre d'orienter le choix. Par exemple, des stratégies spécifiques peuvent être associées au type du choix à réaliser.

Les structures utilisées lors du déploiement peuvent donc être approfondies. Dans ce cas, elles pourront donner plus de possibilités à l'utilisateur. La structure d'entreprise pourra alors être construite selon les besoins de chaque opération de déploiement, et, la structure de plan exprimera davantage de degrés de liberté et d'information sur la manière de les traiter.

2.1.3. Expérimentation

Côté « pratique », quelques éléments doivent être approfondis ou ajoutés. L'implémentation du langage de stratégies permettra alors à l'utilisateur de définir lui-même les stratégies dont il a besoin. En outre, le cas du déploiement de n applications (chacune disponible en m versions) sur p machines permettra de passer au cas le plus général possible.

Une fois ces points ajoutés, une expérimentation industrielle pourra être envisagée. En effet, seule la première version d'ORYA a été expérimentée industriellement. La deuxième version d'ORYA n'a été évaluée qu'avec des tests que nous avons nous-même construits.

L'expérimentation de cette deuxième version en milieu industriel permettra tout d'abord d'évaluer le niveau de difficulté de l'utilisation de stratégies de déploiement. Ainsi, en fonction de ses résultats, des outils pourraient être ajoutés pour faciliter le travail de l'administrateur dans la définition et l'exploitation de stratégies de déploiement.

Une expérimentation en « grandeur réelle » permettra aussi d'évaluer le temps de préparation du déploiement. En effet, pour un grand nombre de machines le temps mis pour sélectionner les unités à déployer et créer et ordonnancer le plan de déploiement doit rester négligeable par rapport au temps du déploiement sur l'ensemble des machines.

2.2. Perspectives d'utilisation

Plusieurs projets en relation avec le déploiement sont en cours dans notre équipe de recherche : ANSO [Anso], S4ALL [S4all], PISE [Pise]. Ces projets réutilisent certains travaux que nous avons présentés dans ce document, éventuellement en les adaptant à un contexte précis.

2.2.1. Déploiement distribué

Un premier objectif est d'ajouter la gestion des applications distribuées. En effet, nous avons traité le cas où la même application est déployée sur plusieurs machines, mais le cas des applications distribuées est différent. Il s'agit alors de déployer une seule application sur plusieurs machines qui coopèrent.

Dans ce cas, lorsque le nombre de machines cibles est très grand, le système devient difficilement gérable. Il est par exemple difficile de définir des stratégies de déploiement pour toutes les machines et de savoir quand appliquer quelle(s) stratégie(s). Il est alors nécessaire de revoir l'architecture du système que nous avons proposée pour ajouter des niveaux intermédiaires.

En ajoutant des niveaux intermédiaires, il s'agit de généraliser la notion de serveur de déploiement. Le déploiement se passe alors à plusieurs niveaux. L'information de déploiement est tout d'abord transmise depuis un producteur jusqu'à une passerelle. Cette information peut être transmise directement (transfert de données) ou indirectement (annonce de la disponibilité avec mention d'une URL).

La passerelle gère ensuite elle-même un ensemble de cibles finales. C'est donc au niveau de la passerelle que le déploiement va être traité. A ce niveau, la décision de distribuer l'application peut, par exemple, être prise. Il peut aussi se produire différentes vérifications de compatibilité entre l'application disponible et les cibles gérées par la passerelle. Un ensemble d'intermédiaires peut alors composer une chaîne de médiation entre le système centralisé et le système terminal (cibles terminales de déploiement).

Deux niveaux de déploiement apparaissent. Un premier déploiement à grande échelle a lieu entre le producteur/fournisseur et les différentes passerelles de ses clients. Dans ce cas, l'utilisation de stratégies, telles que nous les avons décrites, est possible et peut refléter les caractéristiques des contrats signés par les différents clients ou la configuration de leur matériel.

Le deuxième déploiement est géré par la passerelle, vers les cibles finales de déploiement. Par exemple, dans le cas du contexte de services à domicile (projet ANSO), la passerelle représente un point d'entrée dans une maison, et les cibles terminales sont l'ordinateur familial, le home-cinéma, etc. Dans ce cas, le langage de stratégie doit être plus proche de l'utilisateur, tout en permettant d'exprimer les mêmes besoins. A ce niveau, les stratégies permettent de construire un plan de déploiement local, avec éventuellement une partie distribuée.

La décomposition du déploiement en différents niveaux permet aussi de simplifier les cas de chaque niveau. En effet, nous avons vu précédemment l'accroissement de la complexité lorsque le nombre de contraintes et de stratégies de déploiement augmente. Cette

décomposition permet de limiter le nombre de stratégies de déploiement à chaque niveau et de faciliter leur application.

2.2.2. Déploiement de services

Dans notre approche, nous avons utilisé la notion d'unité de déploiement pour abstraire ce qui doit être déployé (principalement des composants, du code, …). Les projets en cours reposent sur le concept généralisé de service. Nous devons donc maintenant étudier ce que signifie « déployer des services ».

En effet, un ensemble de services est fourni dans une même unité de déploiement (bundle, dans OSGi). Ce sont donc des unités qui sont déployées, mais des services qui sont utilisés. Dans ce cas, il faut établir les rapports entre services et unités de déploiement. A l'exécution, le raisonnement se fait en termes de services, alors qu'au déploiement, ce sont des unités qui sont traitées.

Il faut alors déterminer quelles sont les liens entre ces deux notions. Par exemple, il faut définir ce que signifie une dépendance de services : les services dépendants sont-ils fournis par la même unité de déploiement ou des unités différentes ? Le déploiement d'une unité de déploiement peut aussi impliquer le déploiement de services qui ne sont pas nécessaires (mais qui sont contenus dans l'unité déployée), il faut alors déterminer quoi faire de ces services non utilisés. Pour répondre à toutes ces questions, des travaux sont en cours pour définir un modèle de services.

Les perspectives d'utilisation des travaux présentés dans le cadre des projets de notre équipe de recherche s'orientent donc vers le déploiement de services. Ces services sont déployés selon une architecture avec des intermédiaires entre les serveurs centraux et les éléments terminaux (cf. paragraphe 2.2.1). L'architecture de notre système devra être revue pour permettre la gestion de niveaux intermédiaires. Les stratégies de déploiement pourront être utilisées à ces différents niveaux.

BIBLIOGRAPHIE

Préambule :

Cette partie contient l'ensemble des références bibliographiques citées dans ce document.

[ABT+05] D.Ayed , N.Belhanafi, C.Taconet, G.Bernard. “Deployment of component-based applications on top of a context-aware middleware”. The IASTED International Conference on Software Engineering (SE 2005). Innsbruck, Austria. February 15-17, 2005.

[Actoll] Site web de la société Actoll : http://www.actoll.com/

[AM00] G.D.Abowd, E.D.Mynatt. “A Charting past, present, and future research in ubiquitous computing”. CM Transactions on Computer-Human Interaction (TOCHI), v.7 n.1, p.29-58. March 2000.

[And00] R.Anderson. “The End of DLL Hell”. Technical Article of Microsoft Corporation. January 2000.

[Anso] Projet ITEA ANSO n° 04016 : Autonomic Networks for SOHO users.

[ATB03] D.Ayed, C.Taconet, G.Bernard. “Context-Aware Deployment of multi-component applications”. Proceedings of the 5th Generative Programming and Component Engineering (GPCE03), Young Researchers Workshop. Erfurt, Germany. September 2003.

[ATB04] D.Ayed, C.Taconet, G.Bernard. “Architecture à base de composants pour le déploiement adaptatif des applications multi-composants”. Journées Composants 2004. Lillie, France. Mars 2004.

[ATS+05] D.Ayed, C.Taconet, N.Sabri, G.Bernard. “CADeComp : plate-forme de déploiement sensible au contexte des applications à base de composants”. 4ème Conférence Française sur les Systèmes d’Exploitation (CFSE’05). Le Croisic, France. 5-8 Avril 2005.

[BC01] G.Bieber, J.Carpenter. “Introduction to Service-Oriented Programming”. September 2001. http://www.openwings.org/download.html .

[BEM94] N. Belkhatir, J. Estublier, W. Melo. “ADELE-TEMPO : An environment to Support Process Modelling and Enaction”. Book Software Process Modelling and Technology, pages 187-217. John Willey and Son inc, Research Study Press, Tauton Somerset, England. 1994.

[BEM95] N. Belkhatir, J. Estublier, W. Melo. “Process-Centered Software Engineering Environment”. Pankaj K. Garg and Mehdi Jazayeri. ISBN 0-8186-7103-3. 1995.

[BG01] J. Bézivin, O. Gerbé. “Towards a Precise Definition of the OMG/MDA Framework”. ASE’01. November 2001.

[Bou03] C.Boulton. “Service-Oriented Architectures Underpin On-demand”. May 2003. Digital Magazine : InternetNews.com : http://www.internetnews.com/ent-news/article.php/2212131 .

[BTA+05] A.Beloued, C.Taconet, D.Ayed, G.Bernard. "Placement automatique des composants lors du déploiement d'applications à base de composants". Journées Composants (JC'05). Le Croisic, France. 5-8 Avril 2005.

[Car97] D. Carney. "Assembling Large Systems from COTS Components. Opportunities, Caution and Complexities". SEI Monograph on Use of Commercial Software in Government Systems. SEI, Pittsburgh, USA. June 1997.

[Ccm03] CORBA Component Model page. The CORBA Component Model Specification. http://ditec.um.es/~dsevilla/ccm/ .

[CE00] T. Coupaye, J.Estublier. "Foundations of Enterprise Software Deployment". Proceedings of the 4th European Conference on Software Maintenance and Reengineering. Zurick. February 2000. IEEE Computer Society, 2000 pages 65-74.

[CE99] T. Coupaye, J. Estublier. "Entreprise Software Deployment : Foundations & Related Technologies". Rapport de recherche, RR 1023-I-LSR11. Laboratoire LSR, Grenoble, France. Octobre 1999.

[Centr] Site web du projet Centr'Actoll :

http://www-adele.imag.fr/Les.Groupes/centractoll/

[CFH+98] A.Carzaniga, A.Fugetta, R.S.Hall, A.Van der Hoek, D.Heimbigner, A.L.Wolf. "A Characterization Framework for Software Deployment Technologies". Technical Report CU-CS-857-98. Dept. Of Computer Science, University of Colorado, USA. April 1998.

[CL95] R. Conradi, C. Liu. "Process Modelling Languages : One or Many?". In Proceedings of the 4th European Workshop on Software Process Technology. Lecture Notes in Computer Sciences 913, Springer. Noordwijkerhout, The Netherlands. April 1995.

[CLM05] P.Y. Cunin, V. Lestideau, N. Merle. "ORYA : A strategy oriented deployment framework". 3rd International Working Conference on Component Deployment (CD 2005). Grenoble, France. November 28th-29th, 2005.

[CVC+04] L.Charles, M.Vacelet, M.Chaari, M.Santana. "SDS : Une infrastructure d'installation de logiciels libres pour des organisations multi-sites". 1ère conférence francophone sur le déploiement et la (Re)Configuration de logiciels (DECOR'04). Grenoble, France. Octobre 2004.

[DAW98] J.C. Derniame, B. Ali Kaba, D. Wastell. "Software Process : Principles, Methodology, Technology". Lecture Notes in Computer Science 1500. 1998.

[DEA98] S. Dami, J. Estublier, M. Amiour. "APEL : A Graphical yet Executable Formalism for Process Modeling". Kluwer Academic Publishers, Boston. Process Technology. Edited by E. Di Nitto and A. Fuggeta. Pages 61-97. January 1998.

[Der94] J.C. Derniame. “Life Cycle Process Support in PCIS”. Proceedings of PCTE’94. San Fransisco, USA. November 1994.

[ECB98] J. Estublier, P.Y. Cunin, N. Belkhatir. “Architectures for Process Support System Interoperability”. In Proceedings of the 5th International Conference on The Software Process, The International Software Process Association Press, Lisle, IL. June 1998.

[EI05] J. Estublier, A.D. Ionita. “Extending UML for Model Composition”. Australian Software Engineering Conference (ASWEC). Brisbane, Australia. March 2005.

[EIV05] J. Estublier, A. D. Ionita, G. Vega. “A Domain Composition Approach”. Published in 2005 International Workshop on Applications of UML/MDA to Software Systems (UMSS’05) SERP'05. June 2005, Las Vegas, USA.

[EL01] J. Estublier, A.T. Le. “Design and development of Software Federation”. Proceedings of ICSSEA 2001. Paris, France. December 2001.

[Elm90] A.K. Elmagarmid. “Database Transaction Models for Advanced Applications”. Morgan Kaufmann. 1990.

[ES05] J. Estublier, S. Sanlaville. “Extensible Process Support Environments for Web Services Orchestration”. International IEEE Conference on Next Generation Web Services Practices (NweSP’05). Seoul, Korea. August 2005.

[ET96] M. Ewing, E. Troan. “The RPM Packaging System”. In Proceedings of the First Conference on Freely Redistributable Software. Cambridge, MA, USA. February 1996. Free Software Fundation.

[EV05] J. Estublier, G. Vega. “Reuse and Variability in Large Software Applications”. Published in Proc ESEC/FSE. Lisboa, Portugal. September 7-9, 2005.

[EVC01-a] J. Estublier, H. Verjus, P.Y. Cunin. “Designing and Building Software Federations”. 1st Conference on Component Based Software Engineering (CBSE). Varsovie, Poland. September 4-6, 2001. (18/35)

[EVC01-b] J. Estublier, H. Verjus, P.Y. Cunin. “Modelling and Managing Software Federations”. European Conference on Software Engineering (ESEC). Wien, Austria. September 10-14, 2001. (37/137)

[EVC01-c] J.Estublier, H.Verjus, P.Y.Cunin. “Building Software Federations”. The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). Las Vegas, Nevada (USA). June 25-28, 2001.

[EVL+03] J. Estublier, J. Villalobos, Anh-Tuyet. Le, S. Sanlaville, G. Vega. “An Approach and Framework for Extensible Process Support System”. Proceedings of 9th European Workshop on Software Process Technology (EWSPT 2003). Helsinki, Finland. September 2003.

[Fab76] R.S.Fabry. “How to design a system in which modules can be changed on the fly”. Proceedings of the 2nd international conference on Software engineering, p.470-476. San Francisco, California, USA. October 1976.

[Fav04] J.M. Favre. “Towards a Basic Theory to Model Model Driven Engineering”. Workshop on Software Model Engineering, WISME 2004, joint with UML 2004. Lisboa, Portugal. October 11, 2004.

[Gdf05] “On-demand Service Installation and Activation with OSGi”. ObjectWebCon05 : Fourth Annual ObjectWeb Conference. Lyon, France. January 2005.

[Gom00] Ana Laura Gomez. “Modèle de Sites pour le déploiement de logiciels”. Rapport de DEA Systèmes d’Information – MATIS. Université Joseph Fourier, UFR IMA, Grenoble, France. Juin 2000.

[Hal99] R.S.Hall. “Agent-based Software Configuration and Deployment”. PhD Thesis. University of Colorado, USA. 1999.

[Han99] W.J.Hansen. “Deployment Descriptions in a World of COTS ans Open Source”. Proceedings in the International Symposium on System Configuration Management (SCM-9). Lecture Notes in Computer Science 1675, Springer, Heidelberg. September 1999.

[HHC+98] A.Van Der Hoek, R.S.Hall, A.Carzaniga, D.Heimbigner, A.L.Wolf. “Extending Configuration Management Support into the Field”. Crosstalk, The Journal of Defense Software Engneering, volume11, number 2. Febuary 1998.

[HHH+97] R.S.Hall, D.Heimbigner, A.Van Der Hoek, A.L.Wolf. “An Architecture for Post-Development Configuration Management in a Wide-Area Network”. Proceedings of the 17th International Conference on Distributed Computing Systems. Baltimore, USA. May 1997.

[HHW98] R.S.Hall, D.Heimbigner, A.L.Wolf. “Evaluating Software Deployment Languages and Schema”. Proceedings of the 1998 International Conference on Software Maintenance, IEEE Computing Society. November 1998.

[HHW99-a] R.Hall, D.Heimbigner, A.L.Wolf. “A Cooperative Approach to Support Software Deployment Using the Software Dock”. Proceedings of the International Conference on Software Engineering, IEEE Computer Science Press. 1999.

[HHW99-b] R.S.Hall, D. Heimbigner, A.L.Wolf. “Specifying the Deployable Software Description Format in XML”. Technical Report CU-SERL-207-99. Software Engineering Research Laboratory, Department of Computer Science, University of Colorado. March 1999.

[HHW99-c] R.S.Hall, D.Heimbigner, A.L.Wolf. “Enterprise Software Deployment : It’s the Control, Stupid”. Proceedings in the ICSE 99 Workshop “Softawre Engineering over the Internet”. Calgary. May 1999.

[HHW99-d] D.Heimbigner, R.S.Hall, A.L.Wolf. “A Framework for Analyzing Configurations of Deployable Software Systems”. Proceedings of the 5th IEEE International Conference on Engineering of Complex Computer Systems, pages 32-42. Las Vegas, USA. October 1999.

[Ibm] Site web d’IBM : http://www.ibm.com .

[IEV05] A. D. Ionita, J. Estublier, G. Vega. “Domaines Réutilisables Dirigés par les Modèles”. Published in IDM05. Paris, France. July 1, 2005.

[Int] Institut National des télécommunications. Site Internet : http://www.int-evry.fr/ .

[IS11] Outil Installshield 11 de Microsoft. http://www.installshield.com/ .

[Jav02] JAVA. “An abstract model for deployment”. 2002. Disponible à : http://java.sun.com/developer/Books/javaprogramming/jnlp/jnlpch02.PDF.

[Java] Java : java.sun.com .

[JMS+92] M.Jarke, J.Mylopoulos, J.W.Schmidt, Y.Vassiliou. “DAIDA - An environment for evolving information systems”. ACM Transactions on Information Systems, 10, 1. 1992.

[KB03] A.Ketfi, N.Belkhatir. “Dynamic Interface Adapdability in Servcie Oriented Software”. WCOP’03. Darmstadt, Germany. June 2003.

[KBC02] A.Ketfi, N.Belkhatir, P.Y.Cunin. “Automatic Adaptation of Component-based Software : Issues and Experiences”. Proceedings in the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPRA’02). Las Vegas, Nevada, USA. June 2002.

[Ket04] A.Ketfi. “Une approche générique pour la reconfiguration dynamique des applications à base de composants logiciels”. Thèse de doctorat. Université Joseph Fourier, Grenoble, France. Décembre 2004.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.M. Loingtier, J. Irwin. “Aspect-Oriented Programming”. In the Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP’97). Lecture Notes in Computer Science vol. 1241, Springer-Verlag. June 1997.

[LBC00] V. Lestideau, N. Belkhatir, PY. Cunin. “Environnement de déploiement de logiciel automatisé centré entreprise”. ICSSEA 2000-9.

[LBC01] V.Lestideau, N.Belkhatir, P.Y.Cunin. “Un environnement de déploiement automatisé d’applications d’entreprise”. Revue génie logiciel, pages 23-31. Mars 2001.

[LBC02] V.Lestideau, N.Belkhatir, P.Y.Cunin. “Towards automated software component configuration and deployment”. PDTSD’02. Orlando, Florida, USA. July 2002.

[Le04] A.T.Le. “Fédération : une Architecture Logicielle pour la Construction d’Applications dirigée par les modèles”. Thèse de doctorat. Université Joseph Fourier, Grenoble. Janvier 2004.

[Les00] Vincent Lestideau. “Modèle d’application pour le déploiement de logiciel”. Rapport de DEA d’informatique : Communication et coopération dans les systèmes à agents. Université de Savoie, France. Juin 2000.

[Les03] V.Lestideau, “Modèles et environnement pour configurer et déployer des systèmes logiciels”. Thèse de doctorat. Université de Savoie, Annecy. Décembre 2003.

[LEV03] A.T. Le, J.Estublier, J.Villalobos. “Multi-Level Composition for Software Federations”. Proceedings of SC'2003. Warsaw, Poland. April 2003.

[LH05] O. Layaïda, D. Hagimont. “Designing Self-Adaptive Multimedia Applications through Hierarchical Reconfiguration”. 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS). Athens, Greece. June 2005.

[Mar04] C. Marin. “Gestion de la cohérence et des transactions avancées liées au déploiement”. Rapport de DEA. Université Joseph Fourier, Grenoble. Juin 2004.

[MB04-a] N. Merle, N. Belkhatir, “Une architecture conceptuelle pour le déploiement d’applications à grande échelle”. Congrès INFORSID 2004. Biarritz, France. Mai 2004.

[MB04-b] N. Merle, N. Belkhatir, “Open Architecture for Building Large Scale Deployment Systems”. SERP'04. Las Vagas, Nevada, USA. June 2004.

[MBD04] C. Marin, N. Belkhatir, D. Donsez. “Gestion transactionnelle de la reprise sur erreurs dans le déploiement”. 1ère Conférence Francophone sur le Déploiement de la (Re)Configuration de Logiciels. Grenoble, France. Octobre 2004.

[Mer02] N. Merle. “Etude des principes et concepts à la base des architectures pour les systèmes de déploiement”. Rapport de DEA. Université Joseph Fourier, Grenoble. Juin 2002.

[Mer03] N.Merle. “ORYA v1.1 – Manuel d’utilisateur pour le déploiement de la plate-forme Centr’Actoll”. Document technique de l’Equipe Adèle. Décembre 2003.

[Mer04] N. Merle. “Un méta-modèle pour l’automatisation du déploiement d’applications logicielles”. 1ère Conférence Francophone sur le Déploiement de la (Re)Configuration de Logiciels. Grenoble, France. Octobre 2004.

[MH01] V. Marangozova, D. Hagimont. “Availability through Adaptation: a Distributed Application Experiment and Evaluation”. European Research Seminar on Advances in Distributed Systems (ERSADS'2001). Bologna. May 2001.

[MKL+02] N.Merle, A.Ketfi, V.Lestideau, N.Belkhatir. “Spécification du déploiement de la plate-forme Centr’Actoll”. Document technique de l’Equipe Adèle. Novembre 2002.

[Omg03-a] Object Management Group. “MDA Guide Version 1.0.1”. 2003. http://www.omg.org/docs/omg/03-06-01.pdf .

[Omg03-b] Object Management Group. “UML 2.0 OCL Specification”. 2003. http://www.omg.org/docs/ptc/03-10-14.pdf .

[Omg04] OMG, “Deployment and Configuration of Component-based Distributed Applications Specification”. March 2004. http://www.omg.org/docs/ptc/04-03-10.pdf.

[Osd] Open Software Description Format Specification. Disponible à : http://www.w3.org/TR/NOTE-OSD .

[Osg03] Open Services Gateway Initiative. “OSGi Service Platform, specification release 3.0”. March 2003. http://www.osgi/org/ .

[OSK04] M. Oussalah, A. Smeda, T. Khammaci. “An Explicit Definition of Connectors for Component-Based Software Architecture”. Eleventh IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2004), edited by Vaclav Dvorak and Miroslav Sveda. IEEE. 2004.

[Osm05] EUREKA ITEA-Osmose. Site Internet du projet : http://itea-osmose.org .

[PBJ97] F.Plasil, D.Balek, R.Janecek. “DCUP : Dynamic Component Updating in Java/CORBA Environment”. Technical Report No 97/10. Dep. Of SW Engineering, Charles University, Prague. 1997.

[PDC01] A.Parrish, B.Dixon, D.Cordes. “A conceptual foundation for component-based software deployment”. Journal of Systems and Software, Volume 57, Issue 3, pages 193-200. 15 July 2001.

[Pise] Projet RNRT PISE n° 04 R 643 : Passerelle Internet Sécurisée et Flexible.

[Rad] Page web de l’outil Rembo Auto-Deploy. RAD : http://www.rembo.com/products_autodeploy.htm .

[RCl] Techniques de clonage de Rembo. http://www.rembo.com/products_autodeploy_details.htm.

[RDU04] J.L.Ruiz, J.C.Dueñas, F.Usero. “Deployment in dynamic environments”. 1ère conférence francophone sur le déploiement et la (Re)Configuration de logiciels (DECOR’04). Grenoble, France. Octobre 2004.

[RedH] Site Internet de RedHat. http://www.redhat.com/ .

[Rembo] Site Internet des technologies Rembo. http://www.rembo.com/ .

[Rol97] C.Rolland. “A primer for method engineering”. Actes du Congrès INFORSID 1997. Toulouse, France. June 1997.

[Rpm] Site Internet de RPM. http://www.rpm.org/ .

[S4all] Projet ITEA S4ALL n° 04025 : Services FOR ALL.

[San05] S.Jamal-Sanlaville. “Environnement de procédé extensible pour l’orchestration. Application aux services web”. Thèse de doctorat. Université Joseph Fourier, Grenoble, France. Décembre 2005.

[SF93] M.E.Segal, O.Frieder. “On-the-Fly Program Modification : Systems for Dynamic Updating”. IEEE Software, v.10 n.2, pp.53-65. March 1993.

[SOK04] A. Smeda, M. Oussalah, T. Khammaci. “A Multi-Paradigm Approach to Describe Software Systems”. In: WSEAS Transactions on Computers, Issue 4, Volume 3, October 2004, edited by Mastorakis N. WSEAS. 2004.

[SysP] Page web de l’outil SYSPREP de Microsoft. Sysprep : http://www.microsoft.com/technet/prodtechnol/Windows2000Pro/deploy/depopt/sysprep.mspx#EDAA.

[Tcm] Page web du gestionnaire de configuration de Tivoli. Tivoli Configuration Manager : http://www-306.ibm.com/software/tivoli/products/config-mgr/ .

[Tec] Page web de la console d’entreprsie Tivoli. Tivoli Enterprise Console : http://www-306.ibm.com/software/tivoli/products/enterprise-console/ .

[Tivoli] Site web de Tivoli : http://www-306.ibm.com/software/tivoli/ .

[Tlm] Page web de l’outil du gestionnaire de licence de Tivoli. Tivoli License Manager : http://www-306.ibm.com/software/tivoli/products/license-mgr/ .

[TPB03] C.Taconet, E.Putrycz, G.Bernard. “Context Aware Deployment for Mobile Users”. Proceedings of the 27th International Conference on Computer Software and Applications Conference (COMPSAC 2003). IEEE Computer Society. Dallas, USA. November 2003.

[Tsd] Page web de l’outil de distribution logicielle de Tivoli. Tivoli Software Distribution. White Paper : http://www-306.ibm.com/software/tivoli/resource-center/security/wp-software-dist.jsp .

[Upm] Université Polythechnique de Madrid. Site Internet : http://www.upm.es/ .

[Veg05] G.Vega. “Développement d’Applications à grande échelle par composition de méta-modèles”. Thèse de doctorat. Université Joseph Fourier, Grenoble, France. Décembre 2005.

[Vil02] J.Villalobos. “APEL : Spécification formelle du moteur”. Rapport Technique de l’Equipe Adèle. Mars 2002.

[Vil03] J.Villalobos. “Fédération de composants : une Architecture Logicielle pour la Composition par Coordination”. Thèse de doctorat. Université Joseph Fourier, Grenoble, France. Juillet 2003.

[Win] Page web de Windows. http://www.microsoft.com/windows/default.mspx .

[WinI] Service Windows Installer. http://www.microsoft.com/downloads/details.aspx?FamilyID=cebbacd8-c094-4255-b702-de3bb768148f&displaylang=en .

[WMC95] The Workflow Management Coalition Specification. “The Workflow Reference Model”. January 1995. http://www.wfmc.org/standards/standards.htm .

[Xml] World Wide Web Consortium (W3C). “Extensible Markup Language (XML) 1.0, second edition”. Octobre 20002. http://www.w3.org/ TR/REC-xml .

[XmlS] World Wide Web Consortium (W3C). “XML Schema”. http://www.w3c.org/XML/Schema .