HAL
open science

UDeploy : une infrastructure de déploiement pour les applications à base de composants logiciels distribués

Mariam Dibo

To cite this version:

Mariam Dibo. UDeploy : une infrastructure de déploiement pour les applications à base de composants logiciels distribués. Autre [cs.OH]. Université de Grenoble, 2011. Français. NNT : 2011GRENM001 . tel-00685853

HAL Id: tel-00685853

https://theses.hal.science/tel-00685853v1

Submitted on 6 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Mariam DIBO

Thèse dirigée par **Noureddine BELKHATIR**

préparée au sein du **Laboratoire d'Informatique de Grenoble**

dans **l'École Doctorale Mathématiques, Sciences et Technologie de l'Information, Informatique(MSTII)**

UDeploy : Une infrastructure de déploiement pour les applications à base de composants logiciels distribués

Thèse soutenue publiquement le **28/01/2011**, devant le jury composé de :

Prof. Henri Basson,

Université du Littoral, Rapporteur

Prof. Djamal Benslimane,

Université Claude Bernard Lyon, Rapporteur

Prof. Dominique Rieu,

Université Pierre Mendes, Examinateur

Prof. Noureddine Belkhatir,

Université Pierre Mendes, Directeur de thèse

Remerciements

Table des matières

1 L'introduction 13

1.1 Le contexte et la problématique 14

1.2 Le plan du document 15

2 Les concepts clés : composants logiciels et déploiement 17

2.1 Les systèmes à base de composants logiciels 18

2.1.1 Les définitions 18
2.1.2 Les éléments de base 19
2.1.3 Les avantages de l'approche à composants 20

2.2 Le déploiement : définitions, concepts et cycle de vie 21

2.2.1 Les définitions 21
2.2.2 Les éléments de base 22
2.2.3 Le cycle de vie du déploiement 22

3 L'état de l'art et de la pratique 25

3.1 Le déploiement des systèmes monolitiques 27

3.1.1 Software Dock 27
3.1.2 ORYA 31

3.2 Le déploiement dans les intergiciels 33

3.2.1 EJB 33
3.2.2 CCM 37
3.2.3 Canevas .Net 40

3.3 Le déploiement dans la spécification D&C de l'OMG 43

3.3.1 Le modèle d'application 43
3.3.2 Le modèle de domaine 43
3.3.3 Les stratégies de déploiement 43
3.3.4 Le plan de déploiement 43
3.3.5 Synthèse 44

3.4 Le déploiement à haut niveau d'abstraction 45

3.4.1 Fractal 45
3.4.2 SOFA 46
3.4.3 Le déploiement dans les ADL 48
3.4.4 Le déploiement dans UML 50

3.5 L'Étude comparative 51

4 Le cadre conceptuel de notre approche : architecture et meta-modeles 56

4.1 L'architecture 57

4.2 L'approche MDA 60

4.2.1 Les concepts clés 60
4.2.2 Les avantages de l'approche MDA 60
4.2.3 L'approche MDA et le déploiement 60

4.3 Les differents meta-modèles constituant notre système de deploiement 61

4.3.1 Le méta-modèle d'application de UDeploy 61
4.3.2 Le méta-modèle de domaine de UDeploy 65
4.3.3 Le méta-modèle de stratégies de déploiement de UDeploy 69
4.3.4 Le méta-modèle du plan de déploiement de UDeploy 72
4.3.5 Le metamodèle du descripteur de déploiement de UDeploy 73

5 La transformation de modeles et la planification orientée stratégies 76

5.1 La proposition d'un langage de transformation de modèles 77
5.1.1 QVT ATL 78
5.1.2 Algorithmes de transformation 79

5.2 Des exemples de personnalisation sémantique via le QVT ATL 81
5.2.1 La personnalisation du plan de déploiement EJB, .NET et CCM 81
5.2.2 La personnalisation du plan de déploiement SOFA 83
5.2.3 La personnalisation du descripteur de déploiement EJB 85

5.3 Des exemples de personnalisation syntaxique via les algorithmes de transformation 87
5.3.1 La personnalisation du plan de déploiement EJB 87
5.3.2 La personnalisation du plan de déploiement .NET 88
5.3.3 La personnalisation du plan de déploiement CCM 88
5.3.4 La personnalisation du plan de déploiement SOFA 88

5.4 La planification 89

6 La description du prototype 91

6.1 Le prototype UDeploy 92

6.2 L'architecture fonctionnelle 93
6.2.1 La modélisation de l'application, du domaine et des stratégies de déploiement 94
6.2.2 La création du modèle de plan de déploiement 99
6.2.3 La transformation 100
6.2.4 L'exécution du plan de déploiement 104

7 Une étude de cas 107

7.1 La description du cas d'étude 108

7.2 La mise en œuvre du déploiement 112

7.3 Le déploiement du logiciel SCMSoft avec UDeploy 113
7.3.1 Le modèle d'application 113
7.3.2 Le modèle de domaine 114
7.3.3 Les stratégies de déploiement 115
7.3.4 Le modèle de plan de déploiement 116
7.3.5 La transformation 118
7.3.6 L'exécution des modèles de plan de déploiement 121

7.4 L'évaluation technique 122

8 La conclusion et les perspectives 125

8.1 Les contributions 126

8.2 Les perspectives 127

9 La bibliographie 129

10 L'annexe _____ *141*

10.1 Annexe 1 : Algorithme de planification _____ 142

10.2 Annexe 2 : Stratégies de déploiement completes _____ 143

10.3 Annexe 3 : Plan de déploiement complet _____ 145

10.4 Annexe 4 : Publications _____ 147

Liste des figures

FIGURE 1 : L'ARCHITECTURE D'UN COMPOSANT LOGICIEL DU *REPOSITORY* AU CONTENEUR 18
FIGURE 2 : LE PROCESSUS DE DEPLOIEMENT DE SYSTEMES MONOLITHIQUES [CARZANIGA ET AL., 1998] 23
FIGURE 3 : LE MODELE DE DOMAINE DE SOFTWARE DOCK 29
FIGURE 4 : LE META-MODELE ORYA 31
FIGURE 5 : LE META-MODELE D'APPLICATION EJB 33
FIGURE 6 : L'ARCHITECTURE LOGICIELLE ET MATERIELLE D'UNE APPLICATION JAVA EE 35
FIGURE 7 : LE META-MODELE DU PLAN DE DEPLOIEMENT (EJB, CCM, .NET) 36
FIGURE 8 : L'ARCHITECTURE DE DEPLOIEMENT DANS CCM [OMG, 2002] 38
FIGURE 9 : L'ARCHITECTURE LOGICIELLE ET MATERIELLE D'UNE APPLICATION .NET 41
FIGURE 10 : LE META-MODELE DU PLAN DE DEPLOIEMENT SOFA 47
FIGURE 11 : L'ARCHITECTURE DE L'ENVIRONNEMENT UDEPLOY (UNIFIED DEPLOYMENT ARCHITECTURE) 58
FIGURE 12 : LE METAMODELE D'APPLICATION UDEPLOY 62
FIGURE 13 : LE METAMODELE DE DOMAINE UDEPLOY 65
FIGURE 14 : LE META-MODELE DE STRATEGIES UDEPLOY 69
FIGURE 15 : LE META-MODELE DU PLAN DE DEPLOIEMENT UDEPLOY 72
FIGURE 16: LE META-MODELE DU DESCRIPTEUR DE DEPLOIEMENT UDEPLOY 74
FIGURE 17 : LE LANGAGE DE TRANSFORMATION (QVT ET ALGORITHME) 77
FIGURE 18 : LA TRANSFORMATION DE MODELE 78
FIGURE 19 : LA TRANSFORMATION SYNTAXIQUE 79
FIGURE 20 : LA PERSONNALISATION DU PLAN DE DEPLOIEMENT (MM UDEPLOY VERS MM EJB, .NET, CCM) 81
FIGURE 21 : LA PERSONNALISATION DU DESCRIPTEUR DE DEPLOIEMENT (MM UDEPLOY VERS MM EJB) 85
FIGURE 22 : LES ALGORITHMES DE TRANSFORMATION 87
FIGURE 23 : LA CREATION DU PLAN DE DEPLOIEMENT 89
FIGURE 24 : LE DIAGRAMME DE CAS D'UTILISATION DE UDEPLOY 93
FIGURE 25 : L'INTERFACE PRINCIPALE DE UDEPLOY 94
FIGURE 26 : LES FONCTIONS DISPONIBLES POUR LA MANIPULATION DE L'APPLICATION 94
FIGURE 27 : LES FONCTIONS DISPONIBLES POUR LA MANIPULATION DU DOMAINE 94
FIGURE 28 : LES FONCTIONS DISPONIBLES POUR LA MANIPULATION DES STRATEGIES 95
FIGURE 29 : LE DIAGRAMME DE SEQUENCE POUR LA CONNEXION 95
FIGURE 30 : L'INTERFACE GRAPHIQUE POUR LA CONNEXION A LA BASE D'APPLICATION 95
FIGURE 31 : LE DIAGRAMME DE SEQUENCE POUR LA CONSULTATION D'UNE APPLICATION 96
FIGURE 32 : L'INTERFACE POUR LA CONSULTATION D'UNE APPLICATION 96
FIGURE 33: LA CREATION D'UN MODELE D'APPLICATION 97
FIGURE 34 : LA CREATION D'UN COMPOSANT ET SES CONTRAINTES LOGICIELLES ET MATERIELLES 98
FIGURE 35 : LA MODIFICATION D'UNE APPLICATION 99
FIGURE 36 : L'INTERFACE POUR LA MANIPULATION DU PLAN 99
FIGURE 37 : PROCESSUS DE CREATION DU PLAN DE DEPLOIEMENT 100
FIGURE 38 : LA TRANSFORMATION 101
FIGURE 39 : LA FENETRE DE SAISIE DES INFORMATIONS GENERALES 101
FIGURE 40 : LA FENETRE DE SAISIE DES INFORMATIONS D'UN COMPOSANT 102
FIGURE 41 : LA PERSONNALISATION DU MODELE DE PLAN DE DEPLOIEMENT 104
FIGURE 42 : L'EXECUTION DU PLAN 105
FIGURE 43 : L'APPLICATION SCMSOFT 109
FIGURE 44 : LE DOMAINE DE DEPLOIEMENT (ENTREPRISE SOLIS) 111
FIGURE 45 : LE MODELE D'APPLICATION SCMSOFT 114
FIGURE 46 : LE MODELE DE DOMAINE SOLISDOMAINE 114
FIGURE 47 : LE MODELE DE STRATEGIES SOLISSTRATEGIES 115
FIGURE 48 : LE CALCUL DU PLAN DE DEPLOIEMENT (PROCESSUS 1/4 ET 2/4) 116
FIGURE 49 : LE CALCUL DU PLAN DE DEPLOIEMENT (PROCESSUS 3/4) 117
FIGURE 50 : LE CALCUL DU PLAN DE DEPLOIEMENT (PROCESSUS 4/4) 117

FIGURE 51 : L'EVALUATION DE UDEPLOY PAR LA METHODE OSMM 123

Liste des tableaux

TABLEAU 1 : TABLEAU DE COMPARAISON (APPLICATION ET DOMAINE) 54
TABLEAU 2 : TABLEAU DE COMPARAISON (STRATEGIES ET PLAN DE DEPLOIEMENT).....55
TABLEAU 3 : ARCHITECTURE TECHNIQUE DE UDEPLOY 92
TABLEAU 4 : LE MODELE D'APPLICATION SCMSOFT.....109
TABLEAU 5 : LE MODELE DE DOMAINE SOLIS 111
TABLEAU 6 : OSSM ÉVALUATION 122

Résumé

Dans le cycle de vie logiciel nous avons principalement les activités (1) de pré-développement (l'analyse des besoins, les spécifications, la conception architecturale et la conception détaillée), (2) de développement (l'implémentation, le prototypage, les tests unitaires et les tests d'intégration) et (3) de post-développement (déploiement). Le déploiement de logiciel couvre l'ensemble des activités post-développement. Les activités de déploiement permettent de rendre une application utilisable. Elles sont identifiées comme cycle de vie de déploiement couvrant l'archivage des logiciels, leur chargement, leur installation sur les sites clients, leur configuration, leur activation ainsi que leur mise à jour. Le développement de systèmes à composants a permis de mieux identifier cette partie du cycle de vie global du logiciel, comme le montrent de nombreux travaux industriels et académiques. Cependant ces travaux sont en général développés de manière ad' hoc, spécifiques à une plate-forme donnée. Peu flexibles, ils s'adaptent difficilement aux stratégies des entreprises. Les systèmes de déploiement comme le montrent ceux supportés par les environnements de type intergiciel CCM, .Net, EJB développent de manière spécifique les mécanismes et outils de déploiement et introduisent des choix prédéfinis et figés de stratégies de déploiement. Nos travaux se situent dans le contexte de logiciels à base de composants distribués et portent sur la proposition d'un environnement générique pour supporter leur déploiement. C'est une nouvelle génération de systèmes proposée essentiellement par le monde académique de génie logiciel qui s'est approprié la problématique de déploiement à large échelle. Dans ce contexte, nous proposons une approche basée sur l'ingénierie dirigée par les modèles où nous introduisons les abstractions nécessaires pour décrire les logiciels à déployer, les infrastructures de déploiement, les stratégies de déploiement ainsi que le processus de déploiement avec l'identification et l'ordonnancement des activités à accomplir et le support pour leur exécution.

Mots-clés

Déploiement, Application à base de Composants, J2EE, Framework .NET, CCM, SOFA, MDA, Stratégies de déploiement.

Abstract

In the software life cycle we have mainly (1) the pre-development (requirements, specification and design), (2) the development (implementation, prototyping, testing) and (3) the post-development (deployment) activities. Software deployment encompasses all post-development activities that make an application operational. These activities, identified as deployment life cycle, include: i) software packaging, ii) loading and installation of software on client sites, iii) instance creation, iv) configuration and v) updating. The development of system-based components made it possible to better highlight this part of the global software lifecycle, as illustrated by numerous industrial and academic studies. However these are generally developed ad hoc, and consequently platform-dependent. Deployment systems, such as supported by middleware environments (CCM, .Net and EJB), specifically develop mechanisms and tools related to pre-specified deployment strategies. Our work, related to the topic of distributed component-based software applications, aims at specifying a generic deployment framework independent of the target environments. Driven by the meta-model approach, we first describe the abstractions used to characterize the deployed software. We then specify the deployment infrastructure and processes, highlighting the activities to be carried out and the support for their execution.

Keywords

Deployment, Components-based applications, J2EE, .NET Framework, CCM, SOFA, MDA, Deployment strategies.

1
L'INTRODUCTION

1.1 LE CONTEXTE ET LA PROBLEMATIQUE

L'ingénierie des logiciels a connu une évolution importante ces dernières décennies, d'une part due à l'évolution des techniques de développement d'application logiciel en passant par l'approche objet jusqu'à l'approche à composants et, d'autre part due à la multiplicité et à la diversité des plateformes d'exécution (PDA, tablettes PC, téléphones portables).

L'approche à composants et la distribution ont contribué considérablement à l'évolution des tâches manuelles d'administration système vers une automatisation qui cherche à tendre vers zéro administrateur. Cette évolution s'est imposée par le biais de différents domaines nouveaux du génie logiciel qui sont la domotique, les calculs sur grille et l'informatique ambiante. Dans de tels environnements, le déploiement se fait à chaud (au moment où le besoin est exprimé) et se fait en mutualisant les ressources (un logiciel X est désinstallé pour installer un logiciel Y sur un PDA. Dès que l'utilisateur finit d'utiliser le logiciel Y, on réinstalle le logiciel X. Dans de tels cas les logiciels X et Y ne sont pas utilisés simultanément). Des solutions ont été proposées par rapport au déploiement et elles peuvent être classées comme suit : (1) des outils de type installateurs comme InstallShield, Tivoli, (2) des outils d'administration intégrés directement dans les intergiciels comme EJB, CCM, .Net et (3) des outils de planification basés sur les théories de l'intelligence artificielle et qui trouvent leur source dans l'ordonnancement de tâches comme le planificateur SGP (Sensory Graphplan) [Smith and Weld, 1998] [Weld et al., 1998], SHOP (Simple Hierarchical Ordered Planner) [Nau et al., 1999] [Dix et al., 2003], STAN et AltAlt System [Srivastava, 2001].

Les outils de déploiement dédiés pour les applications à base de composants logiciels sont généralement construits de façon ad' hoc et donc sont spécifiques à une technologie donnée [Dibo and Belkhatir, 2009]. L'ensemble des activités d'administration qu'ils couvrent sont appelés déploiement. Ainsi le déploiement est vu comme l'ensemble des activités post-développement qui rendent un logiciel utilisable. Il couvre les activités de description de l'application à déployer, les activités de description des infrastructures matérielles, les activités de planification, les activités d'exécutions de plan et les activités de ré-planification. L'activité de déploiement peut être initiée par le producteur du logiciel ou par le consommateur du logiciel. Dans le modèle *Push*, le producteur décide d'envoyer l'application à ses clients. Il fera soit une notification de l'activité de déploiement à effectuer ainsi le consommateur aura la liberté d'accepter ou de refuser ou soit il informera le consommateur en effectuant l'activité de déploiement sans demander son autorisation. Dans le modèle *Pull*, le consommateur (plateforme d'exécution) décide d'accueillir une application donnée, ce modèle garantit au consommateur une indépendance vis à vis du producteur et une meilleure sécurité pour les applications qui y sont installées.

Ce travail de thèse s'inscrit dans un cadre global de proposition d'environnement de déploiement pour les applications à base de composants logiciels. En effet, nous proposons une nouvelle alternative aux environnements ad' hoc qui mutualise les avantages des différentes approches et tente de résoudre les problèmes inhérents aux différentes approches.

1.2 LE PLAN DU DOCUMENT

Ce document est structuré comme suit :

Le chapire1 présente le contexte global de ce travail. Nous introduisons le sujet d'étude et nous présentons les challenges dans le domaine du déploiement des applications à base de composants logiciels.

Le chapitre 2 présente les concepts fondamentaux de ce travail qui sont les composants logiciels et le déploiement. Nous donnons un ensemble de définitions et d'éléments de base qui caractérisent ces deux concepts. Par la suite, nous présentons les avantages de l'approche à composants, le cycle de vie du déploiement couvrant les activités d'archivage, de désarchivage, d'installation, de désinstallation, de mise-à-jour, d'activation et de désactivation.

Le chapitre 3, présente l'état de l'art et de la pratique. Cette étude nous a permis d'avoir une vision globale des problématiques de déploiement. En effet, l'étude comparative des différentes approches montrent que les approches sont similaires dans leur objectif mais différentes dans leur mode d'implémentation. Ainsi, le bilan tiré de l'état de l'art et de la pratique nous a permis d'arriver à la proposition d'un noyau commun aux différentes approches. Ce noyau commun constitue le cadre conceptuel de notre travail.

Les chapitres 4 et 5 présentent notre approche.

Dans le chapitre 4, nous proposons le cadre conceptuel de notre approche. Nous présentons l'architecture globale de notre environnement de déploiement générique ensuite nous présentons les différents éléments constitutifs de notre système de déploiement. Ces éléments couvrent deux dimensions qui sont le modèle spécifique et le modèle générique.

Dans le chapitre 5, nous proposons un langage de transformation de modèle. En effet, le passage d'un modèle spécifique vers un modèle générique ou le passage d'un modèle générique vers un modèle spécifique se fait nécessairement au travers d'un processus de transformation. Par la suite, nous proposons notre planificateur de déploiement qui :

- prend en entrée des méta-informations relatives à l'application, à l'infrastructure et aux stratégies de déploiement,
- génère un plan de déploiement générique qui sera par la suite personnalisé en plan(s) de déploiement spécifique(s) par transformation de modèles.

Dans le chapitre 6, nous décrivons le prototype UDeploy construit sur la base de cette approche montrant ainsi sa faisabilité sur des intergiciels cibles.

Dans le chapitre 7, nous présentons une étude de cas qui illustre la manière dont l'outil UDeploy permet le déploiement d'applications à base de composants logiciels hétérogènes (CCM, EJB, SOFA).

Enfin nous concluons ce travail en faisant une synthèse des contributions de cette thèse et en dégageant les perspectives de recherche.

2

LES CONCEPTS CLES : COMPOSANTS LOGICIELS ET DEPLOIEMENT

2.1 LES SYSTEMES A BASE DE COMPOSANTS LOGICIELS

2.1.1 Les définitions

Le terme de composant logiciel a été annoncé pour la première fois par Mclorey [McIlroy, 1968] lors de la conférence internationale sur l'ingénierie des logiciels sponsorisée par l'OTAN le 7 octobre 1968 en Allemagne.

Mclorey exprime lors de cette conférence qu'il apparaît nécessaire de produire les logiciels à l'échelle industrielle. Pour cela, il dit qu'il faudrait étudier les techniques de production de masse. Dans ce sens, il avance sa proposition sur les composants comme étant une production de masse ou encore une réplication illimitée d'un prototype. L'idée de Mclorey a mis plus de vingt ans avant d'avoir un essor. Elle devient une réalité industrielle avec les approches à composant comme EJB, CCM, Fractal et Sofa.

Beaucoup de définitions inconciliables ont existé pour le terme de composant logiciel jusqu'en novembre 1999 [Councill et al., 2000]. Les spécialistes ont initié un atelier virtuel pour l'élaboration de définitions pour ce terme et les résultats de cet atelier ont été publiés par Addison Wesley dans [Heineman and Councill, 2001].

Nous présentons dans ce qui suit les principales définitions qui ressortent de ce travail et d'autres définitions plus récentes sur l'approche à composants :

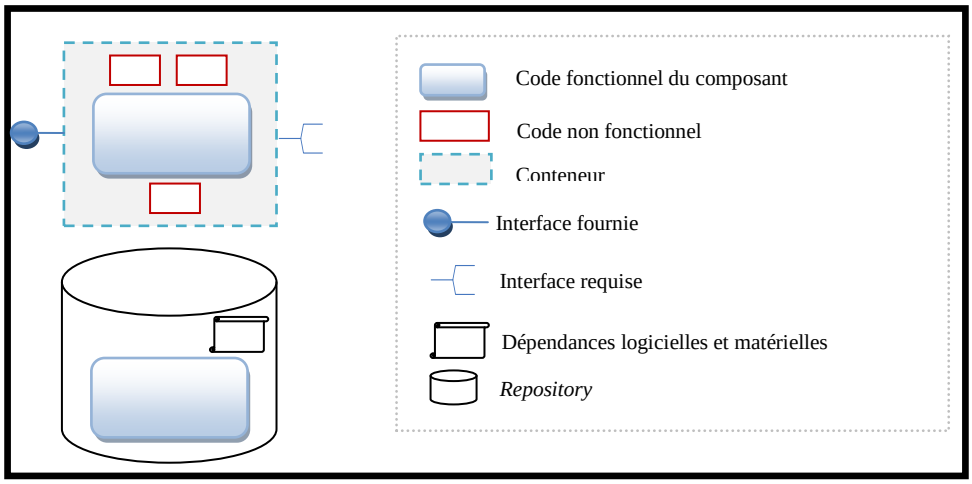


Figure 1 : L'architecture d'un composant logiciel du *repository* au conteneur

- Un composant logiciel exprime les services qu'il offre et qu'il requiert au travers des ports [Councill et al., 2000] (Figure 1). A ces informations s'ajoutent des spécifications sur les dépendances et sur les propriétés non-fonctionnelles du composant [Koziolek and Brosch, 2009].
- Un composant logiciel est une unité logicielle qui peut être déployée de manière indépendante [Szyperski et al., 2002].
- Des composants peuvent être assemblés pour former une application sans modification préalable. C'est le modèle à composants qui détermine les mécanismes d'assemblage [Councill et al., 2000].

- Un composant logiciel est conforme à un modèle à composants, lequel établit la norme pour l'implémentation et l'interopérabilité du composant. En d'autres termes, un modèle à composants régit la manière dont sont construits les composants individuels et dont ils communiquent et interagissent globalement les uns avec les autres [Heineman and Councill, 2001].

2.1.2 Les éléments de base

Dans cette section nous donnons une brève description des concepts clés qui caractérisent l'approche à composants.

Le développement de composants logiciels se fait au travers d'infrastructures qui couvrent les phases d'analyse, de conception, et d'implémentation :

- L'analyse consiste à la modélisation conceptuelle d'un composant du monde réel [Saidi et al., 2008].
- La conception formalise les étapes préliminaires du développement afin de rendre ce développement plus fidèle aux besoins du client [Hoffer et al., 2001] [Hassine et al., 2002] [Ahmad and Basson, 2009].
- L'implémentation est la mise en œuvre du composant conçu [Sharp and Ryan, 2010].

Un composant logiciel une fois implémenté est mis à la disposition des deployeurs via une base de composants. En effet, de nombreux rapports empiriques sur la réutilisation montrent que les utilisateurs à plusieurs reprises reproduisent le même composant et qu'ils ne sont pas conscients que ces composant sont réutilisables [Devanbu et al., 1990] [Fichman and Kemerer, 1997]. Il a été prouvé que réutiliser des composants pouvait apporter un gain en temps et en coût aux industries [Isoda, 1995] [Rosenbaum and du Castel, 1995]. Pour cela, il s'est avéré nécessaire de proposer des bases de composants [Henninger, 1996] [Pan et al., 2004][Qureshi and Hussain, 2008] qui permettent de classifier, de rechercher et de gérer les différentes implémentations et versions des composants.

L'assemblage de composants logiciels [Weide et al., 1991] [Sutcliffe et al., 2006] [Sundarraj, 2002] [Qureshi and Hussain, 2008] se fait soit :

- statiquement. Une configuration logicielle est construite par assemblage de composants disponibles dans la base de composants. Cette configuration doit être valide. Une configuration est valide si et seulement tous les composants mis bout à bout dans une application fonctionnent correctement les uns avec les autres.
- dynamiquement. Une configuration logicielle peut être amenée à évoluer quand le système est en cours d'exécution. Pour cela, un composant est ajouté ou remplacé sans que le système ne soit en arrêt.

2.1.3 Les avantages de l'approche à composants

Les principaux avantages de l'approche à composants sont :

- La réutilisation. L'approche à composants permet le développement de nouvelles applications par assemblage de composants préexistants.
- L'extensibilité. Un composant est extensible sans effets de bord c'est-à-dire que modifier un composant n'a aucun effet sur d'autres composants.
- L'évolution, qui indique la capacité d'un système à évoluer en puissance, le plus souvent par ajout ou remplacement de composants. Ce qui permet à une entreprise d'acquérir un système tout en sachant qu'elle pourra en augmenter les performances éventuellement si le besoin s'en fait sentir (adaptation dynamique ou statique).
- L'indépendance des composants à la compilation c'est-à-dire que la mise-à-jour d'un composant ne nécessite pas la recompilation de toute l'application.
- La programmation et les tests du code sont plus faciles. En effet, la complexité (faible couplage, forte cohésion) d'un composant reste faible par rapport à une application monolithique.
- La fiabilité. Les composants réutilisables sont déjà testés et maintenus. Par conséquent, les composants réutilisables sont plus fiables par opposition à d'autres entités logicielles nouvellement conçues.

Cet intérêt est assujetti à beaucoup de contraintes qui sont :

- Une spécification claire (le comportement, la performance, les interfaces d'entrées et de sorties, la gestion des erreurs).
- Une grande fiabilité (respect des normes du langage, respect des normes de codage, vérification du domaine des paramètres, élimination des pratiques douteuses, testabilité indépendante vis à vis des sorties).
- Une facilité d'intégration (portabilité, respect des normes de nommage, élimination des pratiques empêchant la réentrance, élimination des pratiques empêchant l'instanciation, simplicité des points de sorties).

L'approche à composants consiste alors à créer des entités réutilisables et à développer des logiciels personnalisés grâce à l'assemblage approprié de ces entités. Dans un tel contexte, les rôles des développeurs de composants, des assembleurs d'applications et des deployeurs deviennent clairement différents :

- Les développeurs de composants mettent au point des composants réutilisables.
- Les assembleurs d'applications se concentrent sur le domaine métier en assemblant et en configurant des composants disponibles dans le commerce (le concept de « composants disponibles dans le commerce » devient une réalité grâce aux plateformes de *cloud computing*).
- Les deployeurs d'applications mettent le système logiciel à la disposition des utilisateurs

2.2 LE DEPLOIEMENT : DEFINITIONS, CONCEPTS ET CYCLE DE VIE

2.2.1 Les définitions

Le déploiement de logiciel a peu préoccupé le monde académique qui l'a généralement réduit à la seule activité d'installation de systèmes. L'avènement de systèmes à base de composants distribués a mis en lumière la problématique du déploiement à grande échelle de logiciels constitués de multiples composants à distribuer sur plusieurs sites. Dans un tel contexte, le déploiement est difficilement envisageable sans un support automatisé.

Ainsi, depuis quelques années le monde académique s'est intéressé au déploiement. Les définitions suivantes apparaissent :

- Le déploiement de logiciels est défini comme l'ensemble des activités qui permettent de mettre un logiciel à la disposition des utilisateurs [Carzaniga, 1997].
- Le déploiement de logiciels est une activité complexe et son cycle de vie couvre les activités d'archivage, d'installation, de mise-à-jour et d'adaptation dynamique [Lestideau et al., 2002] [Mikic-rakic and Medvidovic, 2002].

Le déploiement soulève divers aspects, tels que la satisfaction des contraintes logicielles et matérielles des composants par rapport aux ressources des machines qui vont les supporter, la résolution des problèmes de dépendance inter-composants, l'installation et l'instanciation des composants via le *middleware*[1] et le conteneur[2], l'interconnexion des composants, leur activation et la gestion des mises à jour statiques et dynamiques [Dibo and Belkhatir, 2010d].

Le constat suivant résulte de l'ensemble des travaux qui se sont intéressés au déploiement :

- Le déploiement de composants logiciels échoue souvent parce que les dépendances ne sont pas déclarées explicitement. Il en résulte une production incomplète de méta-informations de description nécessaires à la mise en œuvre du déploiement [Dolstra et al., 2004b].
- le déploiement de logiciel peut être un problème déroutant lorsque les composants en cours de déploiement exigent des propriétés non-fonctionnelles qui ne peuvent pas être satisfaites par la plate-forme cible d'exécution [Sommer and Guidec, 2002].

---

[1] « *middleware*» qui se situe entre le système d'exploitation et l'applicatif. Il fait la médiation entre deux systèmes ayant besoin de communiquer entre eux.

[2] Le conteneur supporte le composant. Tout accès au composant se fait par les méthodes produites par le conteneur qui invoque à son tour les méthodes du composant. De ce fait, un client n'a jamais accès directement à un composant.

- Les systèmes existants pour le déploiement de logiciels ne sont ni sûrs ni suffisamment souple. Les questions primaires de sécurité comme le respect des contraintes et le soutien pour des versions multiples de composants ne sont pas souvent prises en compte [Dolstra et al., 2004a].
- Un environnement de déploiement doit être flexible afin de soutenir à la fois le déploiement centralisé de composants, et de permettre différents mécanismes de déploiement spécifiques aux technologies [Dolstra et al., 2004a].
- Le déploiement de logiciels est une partie inévitable du cycle de vie logiciel. Pour les logiciels monolithiques, il existe déjà des outils génériques et des environnements favorisant le déploiement entièrement automatisé. Toutefois, pour les logiciels à base de composants, il n'y a pas d'environnement unifié pour le support des activités de déploiement [Hnetynka, 2004].

Ce constat montre aussi qu'il manque un support automatisé :

- pour la spécification et la résolution des contraintes de placements
- pour rendre flexible le déploiement des composants en offrant des mécanismes pour l'expression des stratégies de déploiement (entreprise et technique)
- qui permet d'unifier le déploiement c'est-à-dire offrir un environnement qui couvre toutes les activités de déploiement.

2.2.2 Les éléments de base

Les unités intervenant dans la constitution d'un système de déploiement sont l'application, le domaine et le plan.

- La notion d'application recouvre l'ensemble des composants de l'application et des méta-informations pour leurs descriptions. Pour chaque composant des contraintes de ressources et des dépendances inter-composants doivent être spécifiées.
- La notion de domaine recouvre l'infrastructure d'exécution des composants. Cette infrastructure est vue comme un ensemble de sites distribués et interconnectés. A chaque site sont associés des méta-informations de description de ses caractéristiques et de son état.
- La notion de plan (procédé de déploiement) définit le processus pour déployer une application totalement ou partiellement. Ce plan de déploiement est guidé par des stratégies de déploiement. Ces stratégies sont généralement définies de manière ad 'hoc.

2.2.3 Le cycle de vie du déploiement

Les premiers travaux [Carzaniga et al., 1998] ont mis en exergue le concept de cycle de vie de déploiement. Le procédé de déploiement est constitué comme le montre la figure 2 d'un ensemble d'activités : l'archivage, le désarchivage, l'installation, la désinstallation, la mise-à-jour, l'activation et la désactivation.

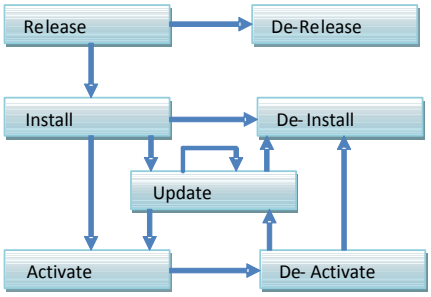


Figure 2 : Le processus de déploiement de systèmes monolithiques [Carzaniga et al., 1998]

- L'archivage qui permet d'assembler correctement un système logiciel. Généralement, cette activité peut être supportée par différents outils comme par exemple Ant [ApacheAnt, 2009], DistAnt [Goscinski and Abramson, 2004] et Maven [ApacheMaven, 2009]. Une archive est vue comme une unité de déploiement.
- Le désarchivage est l'étape où le système devient obsolète c'est-à-dire que le logiciel n'est plus en exploitation.
- L'installation qui concerne la mise en place du logiciel sur un ou plusieurs sites. C'est l'activité centrale du déploiement. Elle est accomplie par des installateurs comme par exemple les outils RPM [RPM, 2010] pour linux et InstallShield [Flexera, 2010] pour windows. Elle doit assurer deux conditions principales qui sont le succès et la sûreté [Parrish et al., 2001] :
    - Succès, l'application à installer s'exécutera conformément aux spécifications du producteur.
    - Sûreté, les applications installées ne seront pas endommagées par la nouvelle installation.
- La désinstallation est l'activité de retrait d'un logiciel.
- La mise-à-jour. Une mise-à-jour peut être statique ou dynamique. Une mise-à-jour statique est une maintenance corrective ou évolutive d'un système en état de désactivation. Une mise-à-jour dynamique aussi appelée adaptation dynamique est une maintenance corrective ou évolutive d'un système qui reste en cours d'exécution donc non-désactivé.
- L'activation qui consiste au lancement de l'exécution.
- La désactivation permet d'arrêter l'exécution d'un système. Elle est souvent nécessaire pour effectuer d'autres activités de déploiement.

Ce cycle de vie du déploiement a été longtemps exploité. Il a servi de modèle de processus pour beaucoup d'outils de déploiement académique comme Software Dock [Hall et al., 1999], Orya [Merle and Belkhatir, 2004] et Dyva [Ketfi et al., 2002]. Ce modèle de cycle de vie a été applicable aux logiciels monolithiques à large échelle et devient adaptable aux logiciels à composant où l'unité de déploiement est à un niveau de granularité plus fine c'est-à-dire le composant [Szyperski et al., 2002].

3

L'ETAT DE L'ART ET DE LA PRATIQUE

Nous avons identifié plusieurs travaux portant sur le déploiement qui ont été classifiés en deux grandes catégories.

Dans la **première catégorie**, il y a tout d'abord ceux plus classiques et développés pour les systèmes logiciels monolithiques qui privilégient l'activité d'installation.

Dans la **seconde catégorie**, il y a tous les systèmes de déploiement développés récemment pour les logiciels à base de composants. Nous avons identifié deux types de systèmes :

- ceux développés par les industriels de manière ad' hoc et intégrés dans des environnements de type intergiciel ;
- ceux de plus haut niveaux d'abstraction à base de modèle explicites proposés d'une part par l'OMG et d'autre part par le monde académique.

3.1 LE DEPLOIEMENT DES SYSTEMES MONOLITIQUES

3.1.1 Software Dock

Software Dock [Hall et al., 1997] est un environnement de déploiement conçu par l'université de Colorado. L'outil est né d'un double constat, le premier est que la frontière entre le déploiement et l'installation n'est pas claire, le deuxième est qu'il existe peu de soutien pour la configuration et le déploiement de logiciel.

a. Le modèle d'application

Dans Software Dock, l'application est modélisée au travers du descripteur DSD. Le DSD « *Description Software Deployment* » est une description déclarative des configurations possibles d'un logiciel. C'est une grammaire DTD « *Document Type Definition* » qui permet de créer des documents XML respectant une sémantique conforme au document type. Il décrit une famille qui est l'ensemble de toutes les versions d'un même logiciel. Cette famille est décrite par cinq classes de propriétés qui sont : l'identificateur, les propriétés, la composition, les contraintes et les artefacts.

– L'identificateur permet de définir des informations sur le nom, la description, le producteur, éventuellement la licence, le logo et la signature du logiciel.

```
<Id>
<Name>LogiChaine</Name>
<Description>Un logiciel de gestion de la chaine logistique</Description>
<Producer>Soft Corporation</Producer>
<License>GNU</License>
<Logo>http://www.Soft.com/logichainelogo.gif</Logo>
<Signature>f2a3b8c32091fd46a785c39723e289a9</Signature>
</Id>
```

– Les propriétés sont de deux types soit externe ou interne. Les propriétés externes sont relatives à la description de la plate-forme d'exécution de l'application. L'OS ou le CPU en sont des exemples. Les propriétés internes décrivent l'application elle-même.

```
<ExternalProperties>
        <ExternalProperty>
                <Name>OS</Name>
                <VarType Value="string"></VarType>
                <Description>Operating system
                </Description>
                <Value>Win32</Value>
        </ExternalProperty>
</ExternalProperties>
```

– La composition permet de définir les relations qu'entretiennent deux ou plusieurs propriétés. Par exemple si la propriété OS est Solaris alors la propriété WinHelp doit

être désactivée (exclusion). Les relations qu'entretiennent les propriétés entre elles sont de quatre types : « *excludes* », « *includes* », « *oneof* » et « *anyof* ». Excludes décrit les relations d'exclusion. Includes décrit les relations d'inclusion. Oneof, vaut vrai si et seulement si la première propriété est vraie et au moins une seule parmi les propriétés suivantes est vraie. Anyof vaut vrai si et seulement si la première propriété est vraie et que toutes les propriétés suivantes sont vraies.

```
        <Composition>
                <Condition>($Online Help$ == true)</Condition>
                <Relation Value="oneof"> </Relation>
                <RuleProperties>
                        <Name>WinHelp</Name>
                        <Name>HtmlHelp</Name>
                </RuleProperties>
        </Composition>
        L'aide en ligne est installée si et seulement si la propriété WinHelp ou
HtmlHelp vaut vraie.
```

– Les contraintes permettent d'exprimer les valeurs des propriétés internes et externes pour que la configuration soit valide. Les contraintes sont de deux types : les assertions et les dépendances.

  - Les assertions décrivent les valeurs valides des propriétés, toutes valeurs choisies en dehors des valeurs définies dans les assertions feront échouer le processus de déploiement. Les assertions sont des contraintes qui sont à valider et ne peuvent pas être résolues au cours du déploiement.

```
        (OS==Win32 || OS==WinNT || OS==Solaris)
```

  - Les dépendances représentent une autre forme de contrainte. Elles peuvent être résolues au cours du processus de déploiement. Les dépendances décrivent l'ordre dans lequel les entités logicielles s'utilisent. Elles permettent par exemple d'installer un logiciel si sa présence est nécessaire. Ainsi le déploiement peut se poursuivre dans de bonnes conditions.

```
        ($Implementation$ == "Java") && (!installed("JVM")) then INSTALL JVM.
```

– Les artefacts décrivent le système comme étant un ensemble de fichiers représentant une famille logicielle. Ils décrivent où sont stockées les applications et où elles doivent être déployées.

```
        <Artifact>
                <Guard>($Implementation$ =="Java")
                </Guard>
                <Signature>96429c4a616df82513e45050ac03d55e</Signature>
                <ArtifactType>CLASSFILE</ArtifactType>
                <SourceName>source.jar</SourceName>
                <Source>/src/java/classes/</Source
                <DestinationName>source.exe</DestinationName>
                …
```

```
        </Artifact>
```

b. Le modèle de domaine

Un domaine de déploiement est vu comme un ensemble de sites interconnectés. Il existe deux types de site dans Software Dock : les sites consommateur et producteur. Un site consommateur est un site potentiel pour accueillir des composants. Un site producteur est un site où se trouve une base de composants. La figure 3 représente le modèle de domaine de software Dock.

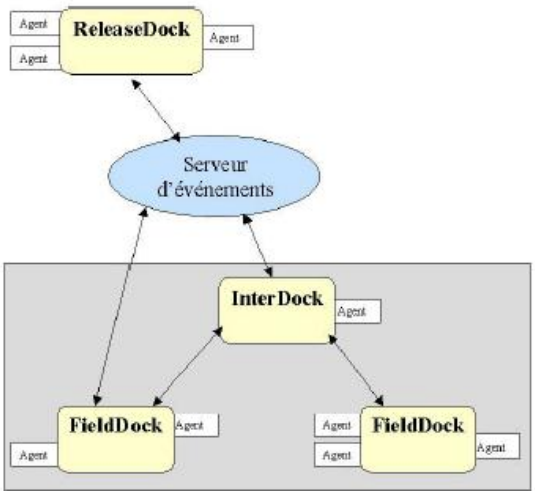


Figure 3 : Le modèle de domaine de Software Dock

Sur chaque site consommateur est installé un *fieldock*. Le *fieldock* offre des informations sur l'environnement logiciel et matériel du site. Il possède des agents de site qui gère les modifications apportées à l'environnement.

Sur chaque site producteur est installé un *releasedock. Le releasedock* offre les informations sur les logiciels disponibles pour le déploiement. Il possède des agents d'installation et de mise à jour. Ces agents peuvent communiquer avec le *fieldock* du site consommateur et le DSD de l'application pour obtenir les informations sur la configuration appropriée.

L'*interDock* est une organisation intermédiaire entre un site producteur et des sites consommateurs.

c. Les stratégies de déploiement

Les stratégies de déploiement dans Software Dock sont figées. Elles guident le choix de la configuration logicielle à déployer.

d. Le plan de déploiement

Le déploiement s'effectue au travers d'un système multi agents.

- L'agent d'installation commence en premier par récupérer le DSD. Le DSD définit un cahier de charge, qui est l'ensemble des configurations valides.

- Ensuite l'agent récupère le *fieldock* du site sur lequel l'application va être déployée. Ce *fieldock* fournit les informations sur le site.
- Zero ou plusieurs configurations peuvent être valides pour le site. Dans le cas où aucune configuration n'est valide pour le site alors le déploiement devient impossible sur le site. Dans le cas où une seule configuration est valide, elle sera déployée. Dans le cas où plusieurs configurations sont valides alors l'utilisateur déterminera la configuration valide à sélectionner.
- Ensuite l'agent résout les problèmes de dépendances, en installant à partir du site producteur les entités utiles à l'exécution du système. L'agent installe aussi tous les artefacts nécessaires au bon emplacement.
- Une fois que les dépendances sont résolues et que les artefacts sont correctement installés, le processus d'installation se termine.

e. Synthèse

Le DSD Offre une forte puissance d'expression. IL permet d'une part, d'exprimer les dépendances, de gérer des contraintes logicielles et matérielles et d'autre part, de définir l'ensemble des configurations valides ainsi que la possibilité d'en choisir une particulière.

3.1.2 ORYA

ORYA (Open enviRonment to deploY Applications) [Cunin et al., 2005] est un environnement pour le déploiement des applications à grande échelle. ORAY a été construit à partir d'outils de déploiement déjà existants. Les auteurs d'ORYA, ont pris le soin de ne pas devenir indépendant d'un ensemble d'outils. Pour cela, ils ont basé leur approche sur un environnement ouvert et évolutif.

a. Le modèle d'application

Le méta-modèle d'application Orya permet de décrire les applications technologiques (EJB, CCM, .NET, Fractal, SOFA), les applications hétérogènes, les propriétés et les contraintes. La granularité reste au niveau application (le produit et les différentes versions) et non au niveau application-composant. La figure 4 représente le méta-modèle d'application pour l'environnement ORYA.

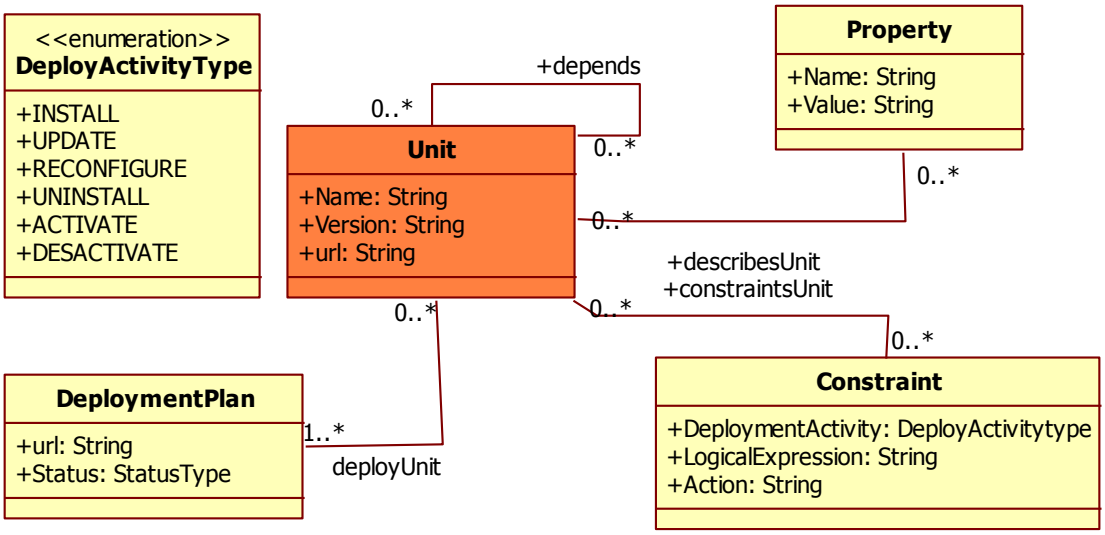


Figure 4 : Le méta-modèle ORYA

Le modèle d'application ORYA présenté ci-dessous décrit un logiciel eclipsewer. Le logiciel eclipsewer est implémenté en Java. C'est un outil de développement. Il peut être installé que sur des systèmes d'exploitation de type winXP ou win2000. Il requiert une capacité mémoire de 104 (l'unité ne peut pas être modélisée via le méta-modèle d'application ORYA).

```
<?xml version="1.0" encoding="UTF-8"?>

<UnitDescriptor xmlns="hhtp://castor.exalab.org/">
   <unitName>eclipsewer</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</propertyName>
        <propertyValue>txt</propertyValue>
      </property>

      <constraint>
        <activity>INSTALL</activity>
        <cExpr>'OS'E {winXP, win2000}</cExpr>
      </constraint>

      <constraint>
        <activity>INSTALL</activity>
        <cExpr>'MEM'>=104</cExpr>
        <action>constraintActionener.substract('Disk',104)</action>
      </constraint>

</unitDescriptor>
```

b. Le modèle de domaine

Le modèle de domaine décrit les différents sites du domaine où les logiciels vont être installés et exécutés ainsi que les ressources logicielles et matérielles offertes par ces sites.

c. Les stratégies de déploiement

ORYA propose un méta-modèle de stratégies qui permet de personnaliser le déploiement en fonction de différents aspects qui peuvent être un ordonnancement particulier, une préférence d'application, une taille mémoire suffisante. Mais, le méta-modèle de stratégies proposé possède peu de sémantique pour exprimer des stratégies techniques (EJB, CCM, .NET).

d. Le plan de déploiement

Le plan de déploiement décrit comment doit s'opérer le déploiement d'une version d'une application. Il peut être unitaire ou à large échelle. Le plan de déploiement unitaire représente une version d'une application qui sera déployée sur une machine. Le plan de déploiement à large échelle représente différentes versions d'une application qui seront déployées sur plusieurs machines.

e. Synthèse

ORYA est orienté vers le déploiement de systèmes monolithiques.

3.2 LE DEPLOIEMENT DANS LES INTERGICIELS

3.2.1 EJB

a. Le modèle d'application

Une application est constituée d'assemblages de composants écrits en Java qui représentent des unités de déploiement. L'architecture applicative est à trois niveaux illustrée par la figure 5. Un assemblage peut être l'implémentation d'un composant métier, web ou client.

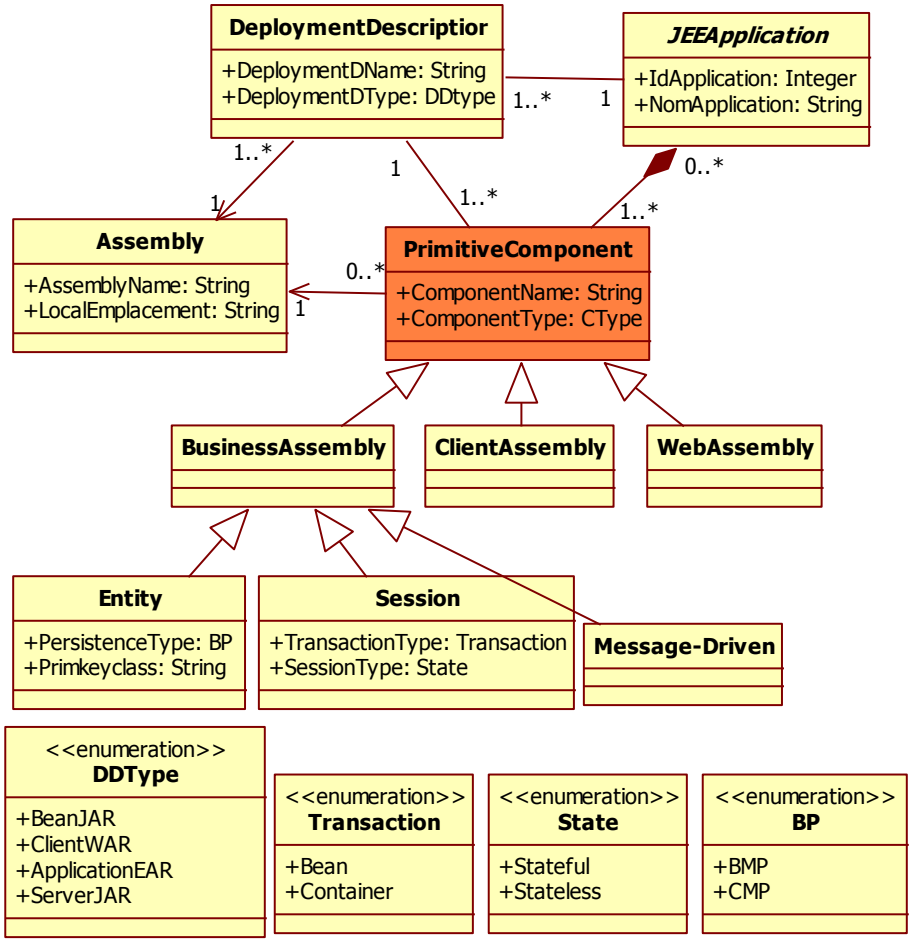


Figure 5 : Le méta-modèle d'application EJB

– Un assemblage métier est une archive qui contient un ou plusieurs composants *enterprise beans*. Chaque assemblage métier est décrit par un descripteur. Ce descripteur contient des informations de base sur les composants comme par exemple le type de composant (entité, session, message), la transaction, la persistance et la sécurité [Keith and Schincariol, 2006] et il ne contiendra aucune information sur les besoins logiciels et matériels du composant.

– Un assemblage Web est une archive constitué de composants web. Les composants Web gèrent les entrées utilisateurs et envoient ces entrées aux composants du niveau métier.

– Un assemblage client est une archive qui contient une application cliente ou un client Web.

Le modèle d'application EJB présenté ci-dessous décrit une application composée de deux composants *EmployeService* et *Employe*. Le composant *EmployeService* est de type *stateful session* et sa transaction est gérée par le Bean. Le composant *Employe* est de type *entity* et sa persistance est gérée par le Bean. Nous ne pouvons pas exprimer des contraintes sur les ressources matérielles et logicielles alors que les contraintes des composants *EmployeService* et *Employe* peuvent être une version spécifique d'un logiciel ou une certaine taille mémoire suffisante.

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
   <enterprise-beans>

      <session>
         <ejb-name>EmployeService</ejb-name>
         <home>com.wombat.empl.EmployeServiceHome</home>
         <remote>com.wombat.empl.EmployeService</remote>
         <ejb-class>com.wombat.empl.EmployeServiceBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Bean</transaction-type>
      </session>

      <entity>
         <ejb-name>Employe</ejb-name>
         <home>com.wombat.empl.EmployeHome</home>
         <remote>com.wombat.empl.Employe</remote>
         <ejb-class>com.wombat.empl.EmployeBean</ejb-class>
         <persistence-type>BMP</persistence-type>
         <prim-key-class>numero</prim-key-class>
      </entity>

      …
   </enterprise-beans>
</ejb-jar>
```

b. Le modèle de domaine

L'environnement de déploiement des applications peut être vu comme une architecture à quatre niveaux (client, Web, métier, *EIS-Enterprise Information System*) :

– Les sites clients correspondent aux terminaux utilisateurs. C'est l'environnement direct de travail des personnels d'entreprise.

– Le serveur web est l'environnement d'exécution des composants Web.

– Le serveur Java EE est l'environnement d'exécution des composants métiers.

– Le serveur de base de données correspond au système d'information de l'entreprise. Ces données peuvent être enrichies, améliorées, utilisées et partagées par plusieurs logiciels.

c. Les stratégies de déploiement

Nous avons identifié trois stratégies de déploiement relatives au placement des composants illustrées par la figure 6 et une stratégie de déploiement relative à la mise-à-jour des composants.

- les assemblages métiers : les assemblages métiers doivent être déployés sur des serveurs dans des conteneurs EJB (obligatoire). La persistance des composants métiers est souvent faite dans une base de données, par défaut le serveur de base de données est celui du serveur d'application Java EE qui héberge les composants métiers (par défaut).
- les assemblages Web : les assemblages Web doivent être déployés sur des serveurs Java EE dans des conteneurs Web (obligatoire).
- les assemblages clients : les assemblages clients doivent être installés sur des machines clientes c'est-à-dire sur des postes d'utilisateurs finaux ou sur des serveurs accessibles aux utilisateurs finaux (obligatoire).
- la Mise à jour : Par défaut, installer un composant x sur un serveur où un autre composant x est déjà déployé consiste à remplacer le composant x déjà installé par celui à installer quelque soit la version de l'un ou de l'autre (par défaut).

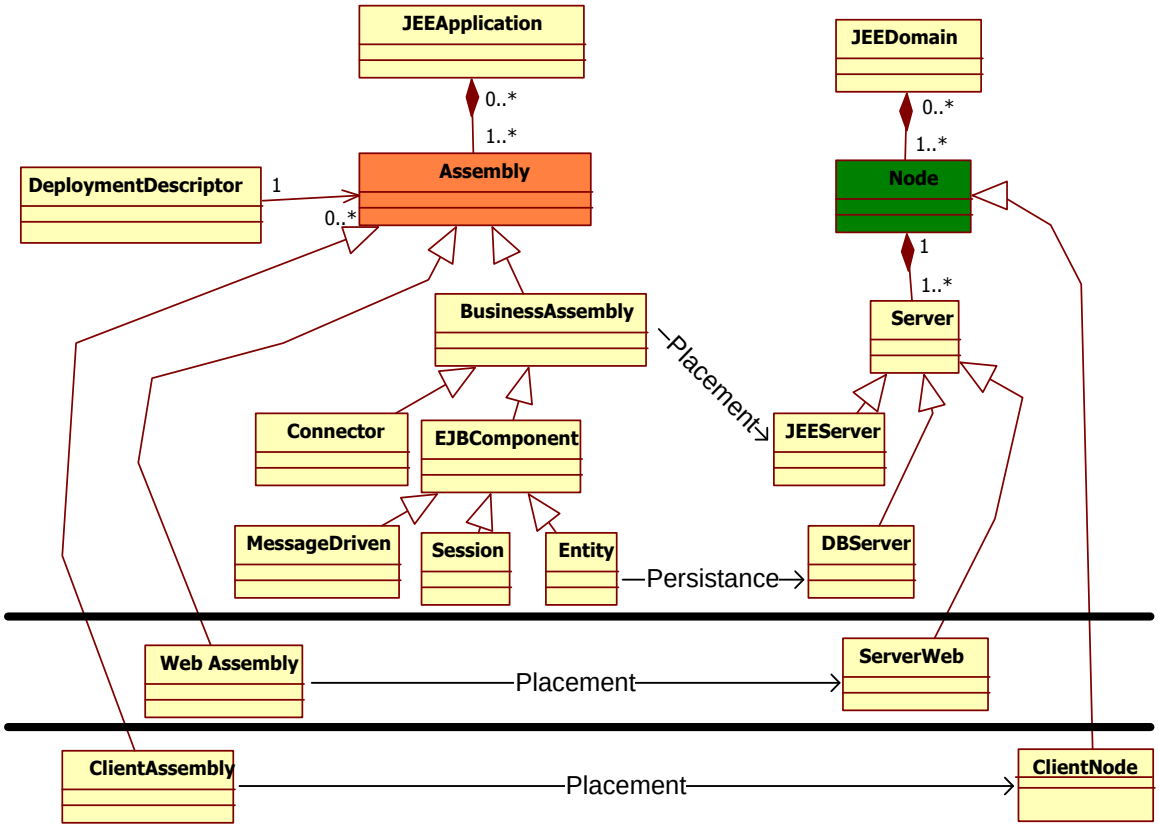


Figure 6 : L'architecture logicielle et matérielle d'une application Java EE

d. Le plan de déploiement

Le plan de déploiement décrit comment une application ou un composant peut être déployé sur une infrastructure. Ce plan de déploiement ne permet que de décrire un déploiement centralisé c'est-à-dire qu'il ne décrit que le déploiement sur un seul site. Le plan de déploiement pour les approches à composants comme EJB, CCM, et .NET sont unitaires c'est-à-dire qu'un plan de déploiement est formé d'un ensemble de sous plans de déploiement (script). Chaque sous plan de déploiement est lié à un seul site. Formellement, il n'existe pas un méta-modèle de plan de déploiement pour ces approches. Nous proposons dans la figure 7, un méta-modèle qui conceptuellement permet de décrire le concept de plan de déploiement dans les approches à composants (EJB, CCM et .NET).

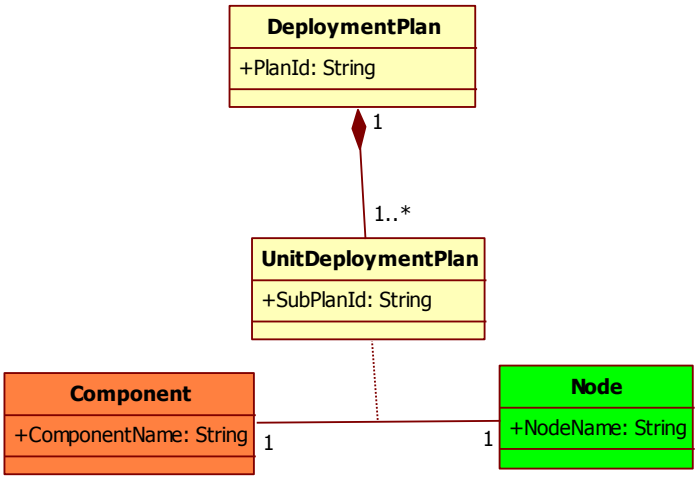


Figure 7 : Le méta-modèle du plan de déploiement (EJB, CCM, .NET)

Le modèle de plan de déploiement EJB ci-dessous présente les différents placements à effectuer afin de déployer une application EJB. Les composants article-EJB-v3.jar et commande-EJB-V3.jar seront installés respectivement sur les nœuds H1 et H2 offrant un serveur d'application JBOSS. La syntaxe aurait été différente si la plateforme d'exécution était Jonas au lieu de JBOSS.

```
DeploymentPlanModel EJBDP1 is

   DeploymentSubPlanModel EJBDSP1 On Node H1 is
     twiddle invoke "jboss.system:service= MainDeployer" deploy file:article-EJB-v3.jar

   DeploymentSubPlanModel EJBDSP2 On Node H2 is
    twiddle invoke "jboss.system:service= MainDeployer" deploy file:commande-EJB-v3.jar
```

e. Synthèse

Dans EJB, il n'existe pas de descripteur de domaine pour décrire l'architecture matérielle.

Le descripteur d'application ne permet pas de décrire des informations plus élaborées comme les besoins en ressources matérielles et logicielles des composants.

Le processus de déploiement est enfoui et spécifique à chaque environnement cible (JBoss ou Jonas).

3.2.2 CCM

a. Le modèle d'application

Le modèle à composant corba (CCM) propose quatre types de composants (*service, session, process et entity*). Les composants sont interconnectés entre eux au travers de ports (*facets, receptacles, event source, event sink*). Ils sont décrits par le descripteur de composant CORBA et par le descripteur de propriétés. Le descripteur de composant CORBA spécifie les paramètres de configuration du composant qui sont évalués pendant le déploiement et qui permettent de changer les comportements d'un composant. Le descripteur de propriétés spécifie le mécanisme de personnalisation d'un composant développé prêt à être réutilisé dans différents scénarios par l'instanciation des attributs à une valeur particulière. Ces deux descripteurs sont spécifiés par le développeur de composant.

L'unité de déploiement peut être un assemblage de composants ou un paquetage d'assemblages :

- L'assemblage de composant contient l'implémentation dans un langage spécifique généralement en C++ ou Java d'un composant ou d'un ensemble de sous-composants interconnectés. L'assemblage de composant est construit par la fabrique de composant et est décrit par le descripteur d'assemblage de composant. Le descripteur d'assemblage de composant spécifie comment les composants et les sous-composants peuvent être instanciés.
- Le paquetage d'assemblages est constitué d'un ensemble d'assemblages d'un même composant dans différentes implémentations. Le paquetage d'assemblages est décrit au travers du descripteur de paquetage logiciel qui spécifie les dépendances vers des fichiers et les dépendances vers des systèmes (OS, ORB).

b. Le modèle de domaine

L'architecture matérielle n'est pas décrite dans la spécification CCM [OMG, 2002]. L'environnement de déploiement est vu comme un ensemble de sites interconnectés disposant de suffisamment de ressources matérielles et logicielles pour accueillir des composants logiciels.

c. Les stratégies de déploiement

Nous avons identifié quatre stratégies de déploiement dans CCM qui sont :

- Les stratégies à l'installation, le descripteur du paquetage logiciel exprime les dépendances fortes (*Assert*) et les dépendances faibles (*Install*). Les dépendances fortes ne peuvent pas être résolues à l'installation (OS, ORB). Quant aux dépendances faibles, elles peuvent être résolues à l'installation.
- Les stratégies de partitionnement, le descripteur d'assemblage de composant exprime comment les assemblages doivent être partitionnés : déploiement particulier d'un composant ou groupe d'assemblages de composants qui doivent être déployés ensemble sur un seul site.

– Les stratégies de configuration, le descripteur de composant CORBA spécifie les paramètres de configuration du composant qui sont évalués pendant le déploiement et qui permettent de changer les comportements d'un composant.

– Les stratégies de personnalisation, le descripteur de propriétés spécifie le mécanisme de personnalisation d'un composant développé prêt à être réutilisé dans différents scénarios par l'instanciation d'attributs.

d. Le plan de déploiement

Le déploiement est mis en œuvre dans CCM par plusieurs interfaces comme le montre la figure 8 : les interfaces *ComponentInstallation, AssemblyFactory, Assembly, ServerActivator, ComponentServer* et *Container*.

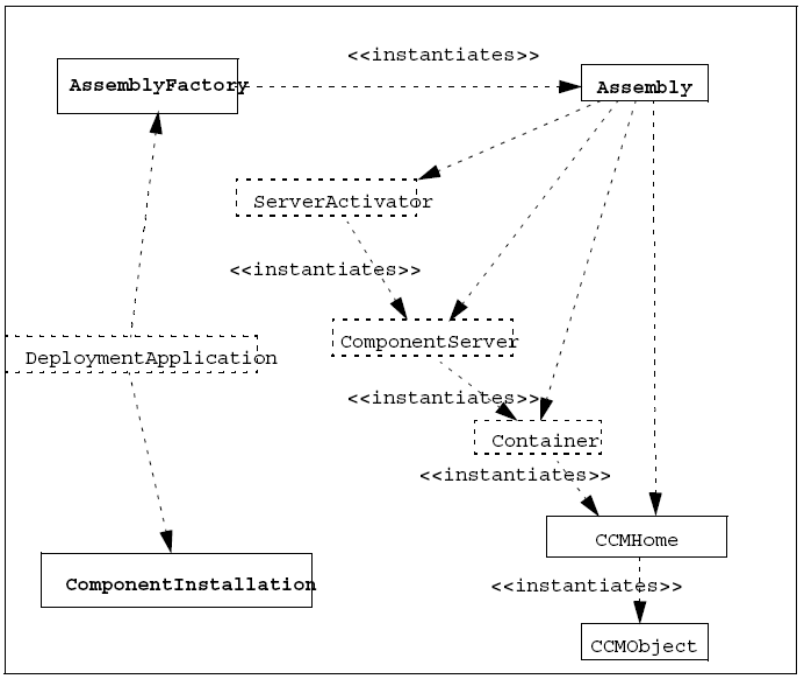


Figure 8 : L'architecture de déploiement dans CCM [OMG, 2002]

– L'interface ComponentInstallation, est utilisée pour installer, retrouver et supprimer les assemblages de composant sur un nœud unique. Il ne doit y avoir qu'un objet *ComponentInstallation* par nœud.

– L'interface *AssemblyFactory*, est utilisée pour créer des instances d'assemblage de composants. Un objet unique d'*AssemblyFactoy* doit être présent sur chaque nœud où les assemblages de composants sont créés. Au moins un objet *AssemblyFactory* doit exister sur un domaine de déploiement.

– L'interface *Assembly*, représente une instance d'assemblage de composant déployée. Il permet de contrôler l'instanciation de l'assemblage ainsi que sa destruction.

– L'interface *ServerActivator* représente le gestionnaire de serveurs d'applications et fournit les opérations pour créer, retrouver et détruire des serveurs. Il ne doit y avoir qu'une instance de *ServerActivator* par nœud.

– L'interface *ComponentServer* représente un serveur d'applications et fournit les opérations pour créer, retrouver et détruire des conteneurs.

– L'interface *Container* représente un conteneur pouvant accueillir des maisons de composants et fournir les opérations pour installer, retrouver et détruire ces maisons.

Le modèle de plan de déploiement CCM ci-dessous présente l'ensemble des placements à effectuer afin de déployer une application CCM. Les composants article-CCM-v3.jar et commande-CCM-V3.jar seront installés respectivement sur les nœuds H1 et H2.

```
DeploymentPlanModel CCMDP1 is

   DeploymentSubPlanModel CCMDSP1 On Node H1 is
      install (article-CCM-v3.jar)

   DeploymentSubPlanModel CCMDSP2 On Node H2 is
     install(commande-CCM-v3.jar)
```

e. Synthèse

Chaque intergiciel CCM redéveloppe de manière spécifique les standards de l'architecture CCM décrit ci-dessus comme par exemple OpenCCM [OpenCCM, 2010] du LIFL, MicoCCM [MicoCCM, 2010] et K2CCM.

Dans CCM, la création du plan de déploiement n'est pas automatisée. C'est l'administrateur système qui construit le plan de déploiement (projection des composants sur les sites).

3.2.3 Canevas .Net

a. Le modèle d'application

Une application .Net [Lantim, 2003] est constituée d'assemblages qui représentent les unités de déploiement. Un assemblage de composant peut être implémenté dans différents langages (Visual Basic, Perl .NET, Cobol .NET, C++/CLI). L'assemblage est un exécutable ou un *assembly*. Un assemblage peut être privé ou public [Troelsen, 2008a, Troelsen, 2008b] :

- Les assemblages publics peuvent être partagés entre plusieurs applications. Tout assemblage public a un nom fort. Le nom fort permet d'identifier de façon unique l'assemblage.
- Les assemblages privés n'ont pas un nom fort et ils peuvent cohabiter à plusieurs. De ce fait, pour éviter le partage de mauvaise version de dll qui est un problème récurant dans .NET, ils ne seront pas partagés.

Chaque assemblage .NET est décrit par un descripteur de déploiement. Ce descripteur est le manifeste qui contiendra les informations suivantes : nom de l'assemblage, le type d'assemblage, numéro de version, langage, clé publique, liste de tous les fichiers de l'assemblage (code et données), liste des références statique sur d'autres assemblages et les stratégies de mise-à-jour.

b. Le modèle de domaine

L'architecture de la plateforme .NET peut être vue comme étant une architecture client-serveur [Lantim, 2003]:

- Les serveurs correspondent aux environnements d'exécution des assemblages publics.
- Les sites clients correspondent aux terminaux utilisateurs. C'est l'environnement direct de travail des personnels d'entreprise où s'exécutent les assemblages privés.

c. Les stratégies de déploiement

Nous avons identifié deux stratégies de déploiement relatives au placement des assemblages illustrées par la figure 9 et des stratégies de déploiement relatives à la mise-à-jour :

- Les stratégies de déploiement des assemblages partagés : les assemblages publics doivent être déployés que sur des serveurs GAC (obligatoire) [Smacchia, 2003].
- Les stratégies de déploiement d'assemblages privés : les composants privés doivent être installés que sur des sites clients disposant d'un environnement d'exécution de ces composants (obligatoire) [Smacchia, 2003].

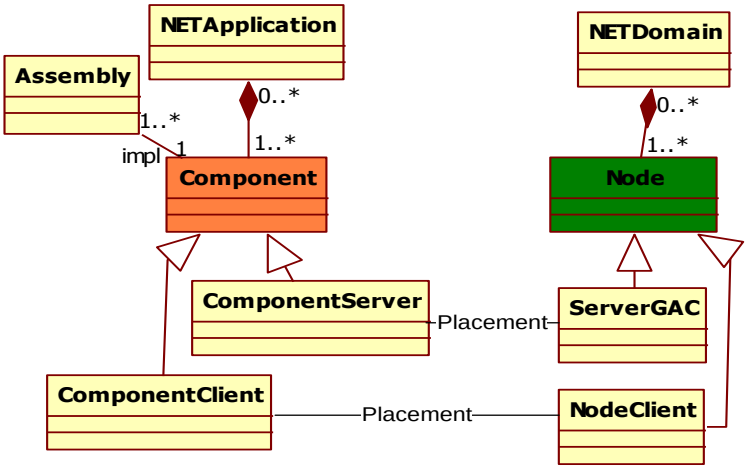


Figure 9 : L'architecture logicielle et matérielle d'une application .NET

– Les stratégies de mise à jour : par défaut, installer un composant x sur un serveur GAC où un autre composant x est déjà déployé consiste à donner un nom fort à ce nouveau composant (par défaut). Les mises à jour peuvent être planifiées par :

  - Une vérification des mises à jour après le démarrage de l'application (*maximumAge* est le nombre de fois où la vérification peut être effectuée et *Unit* est la périodicité).

```
<!-- When to check for updates -->
<subscription>
        <update>
                <expiration maximumAge="6" unit="hours"/>
        </update>
</subscription>
```

La mise à jour sera proposée jusqu'à 6 fois par jour à chaque activation de l'application

  - Une vérification des mises à jour avant le démarrage de l'application

```
<!-- When to check for updates -->
<subscription>
        <update>
                <beforeApplicationStartup />
        </update>
</subscription>
```

L'application tente de localiser le fichier manifeste de déploiement à chaque fois que l'application est activée. Si une mise à jour est disponible, elle sera téléchargée et lancée ; sinon la version existante de l'application sera activée

  - Une activation de l'option de mise à jour obligatoire.

```
<deployment install="true" minimumRequiredVersion="1.0.2.0">
```

Les modifications des propriétés d'un environnement d'exécution peuvent empêcher le fonctionnement correct d'une version spécifique d'un composant. Dans ce cas, la mise à jour du composant est marquée comme étant obligatoire

d. Le plan de déploiement

Dans le canevas .NET comme dans EJB et CCM, le plan de déploiement est unitaire et ne concerne qu'un seul site.

Le déploiement est fait en appelant les méthodes de l'intergiciel (*Global Assembly Cache)*. Le modèle de plan ci-dessous décrit l'ensemble des placements à effectuer afin de déployer une application .NET. Les composants article-NET-v3.exe et commande-NET-V3.dll seront installés respectivement sur les nœuds H1 et H2.

```
DeploymentPlanModel NETDP1 is

   DeploymentSubPlanModel NETDSP1 On Node H1 is
      gacutil -i article-NET-v3.exe
       < !-- gacutil -i, installe un assembly dans le Global Assembly Cache -->


   DeploymentSubPlanModel NETDSP2 On Node H2 is
      gacutil -if commande-NET-v3.dll
       <!-- gacutil -if, installe un assembly dans le Global Assembly Cache. Si un
assembly portant le même nom existe déjà dans le Global Assembly Cache, l'outil le
remplace -->
```

e. Synthèse

Nous retenons du déploiement dans.NET que :

- Le descripteur d'application (manifeste) ne permet pas de décrire les besoins en ressources des composants de l'application.
- Il n'existe pas de description pour l'environnement cible d'exécution des composants.
- Le calcul et la génération du plan de déploiement ne sont pas supportés par le canevas .NET.

3.3 LE DEPLOIEMENT DANS LA SPECIFICATION D&C DE L'OMG

3.3.1 Le modèle d'application

La spécification D&C [OMG, 2006b] propose deux modèles d'application. Le premier modèle représente les méta-informations de description d'une application et le deuxième modèle représente le modèle de processus pour la manipulation des données.

Le modèle de données d'application permet de décrire l'application. Une application est constituée d'un ou de plusieurs composants. Les composants logiciels sont vus comme du code compilé (implémentation monolithique), ou comme un assemblage d'autres composants monolithiques. Les implémentations des composants monolithiques sont décrites en exprimant les contraintes logicielles et matérielles qu'ils requièrent.

Le modèle de processus d'application permet de gérer l'installation d'assemblage de composant à partir de l'URL vers le référentiel (base de données).

3.3.2 Le modèle de domaine

Le modèle de données de l'environnement cible permet de décrire l'infrastructure de déploiement. L'environnement cible est composé de nœuds, d'interconnexions et de ponts. Les nœuds, les interconnexions et les ponts sont décrits en précisant les ressources logicielles et matérielles qu'ils offrent.

Le modèle de processus de l'environnement cible permet de gérer le domaine c'est-à-dire de fournir des informations sur les nœuds et les ressources disponibles et de réserver les ressources pendant l'activité de planification.

3.3.3 Les stratégies de déploiement

La spécification D&C ne permet pas d'exprimer des stratégies de déploiement. La configuration des propriétés non fonctionnelles est prédéfinie et seule la configuration des propriétés fonctionnelles est possible.

3.3.4 Le plan de déploiement

Le plan de déploiement est calculé pendant le processus de déploiement. Le processus de déploiement commence après que le logiciel soit mis au point et publié par un fournisseur. Le processus de déploiement dans D&C est constitué de cinq activités qui sont l'installation, la configuration, la planification, la préparation et le lancement.

- L'installation consiste à transférer le paquetage contenant les implémentations des composants sur un *référentiel* mais l'endroit où se trouve ce *référentiel* n'est pas forcément celui où l'application doit réellement être exécutée.
- La configuration permet de configurer les différents composants. Les modifications ne concerneront pas les propriétés non-fonctionnelles qui sont gérées à partir du

conteneur ou du descripteur de déploiement mais plutôt les propriétés fonctionnelles des composants.

- La planification exprime comment et où l'application doit être exécutée. Cette activité tient compte des exigences du logiciel à déployer et des ressources offertes par l'environnement d'accueil des composants. Un plan de déploiement est produit à l'issue de l'activité de planification.
- La préparation consiste à effectuer un ensemble de tâches pour permettre à l'environnement cible de pouvoir exécuter l'application. Cette étape correspond au déploiement effectif des composants c'est-à-dire le transfert des implémentations binaires de composants depuis le référentiel vers les sites d'exécution.
- Le lancement correspond à l'activation de l'application c'est-à-dire qu'elle devient exécutable.

Dance (Deployment And Configuration Engine) [Deng et al., 2005] est une implémentation de cette spécification. Elle traite de la configuration et du déploiement des applications à base de composants. Elle emploie des descripteurs de déploiement XML pour conduire le déploiement qu'elle effectue. Actuellement, Dance soutient seulement le déploiement et la configuration des applications à base de composants du modèle CCM

3.3.5 Synthèse

L'adaptation de la spécification D&C à d'autres approches à composants est difficilement envisageable, car le méta-modèle d'application proposé dans D&C est très proche de la vision CORBA.

3.4 LE DEPLOIEMENT A HAUT NIVEAU D'ABSTRACTION

3.4.1 Fractal

a. Le modèle d'application

Fractal [Fractal, 2009] est un modèle à composants hiérarchiques. L'unité de déploiement est l'assemblage de composant. Le langage Fractal ADL permet de décrire, à l'aide d'une syntaxe XML, l'architecturelle logicielle d'une application.

b. Le modèle de domaine

Fractal n'impose pas une architecture matérielle. Il propose des intergiciels qui doivent être installés sur les environnements cibles d'exécution. Il n'existe pas de modèle de domaine dans Fractal décrivant les ressources logicielles et matérielles disponibles.

c. Les stratégies de déploiement

Nous avons identifié deux stratégies de déploiement dans Fractal.

- Les stratégies techniques (obligatoire), les composants Fractal doivent être déployés sur des intergiciels cibles conformes au standard Fractal (Proactive est un intergiciel pour les grilles de calcul, Petals est un intergiciel pour les applications d'entreprise/EAI et Frascati est un intergiciel pour les architectures orientées services).
- Les stratégies de mise à jour (par défaut), la mise à jour se fait au travers de contrôleurs qui permettent d'adapter les applications à différents contextes d'exécution (par exemple avec des ressources plus ou moins contraintes).

d. Le plan de déploiement

La notion de plan de déploiement n'existe pas dans Fractal. Fractal utilise des API pour procéder au déploiement.

e. Synthèse

Fractal offre un cadre complet pour l'implémentation, le déploiement et le monitoring d'applications respectant le standard Fractal. Le processus de déploiement faisant appel à des API spécifiques à Fractal reste difficilement applicable à d'autres approches à composants.

3.4.2 SOFA

SOFA (SOFtware Appliances) [Bures et al., 2006] est un modèle à composants développé à l'Université Charles (Prague). SOFA est un modèle à composants hiérarchique supportant l'adaptation dynamique des composants.

a. Le modèle d'application

Un composant SOFA peut être primitif ou composite. L'unité de déploiement dans SOFA est l'assemblage de composant. L'assemblage peut être implémenté dans différents langages mais bien évidemment en accord avec les standards SOFA. Le standard impose deux parties dans un composant : la partie permanente et la partie remplaçable. La partie permanente du composant ne peut pas être mise à jour contrairement à la partie remplaçable. Le modèle d'application dans SOFA ne permet pas de définir les contraintes logicielles et les matérielles des composants.

b. Le modèle de domaine

Dans SOFA le domaine est composé d'un ensemble de sites appelés Dock et d'un site principal appelé SOFAnode. Le déploiement se fait de façon centralisée sur le nœud SOFAnode et les machines (Dock) instancient à distance les composants. SOFAnode est enrichi par l'infrastructure DCUP qui permet de supporter l'adaptation dynamique de la partie remplaçable des composants. Il n'y a pas de modèle de domaine décrivant ce domaine.

c. Les stratégies de déploiement

Nous avons identifié deux stratégies de déploiement dans SOFA.

- Les stratégies techniques (obligatoires), les composants SOFA doivent être déployés sur un site SOFAnode.
- Les stratégies de mise à jour (par défaut), la mise à jour statique et dynamique se fait au travers de DCUP [Plasil et al., 1998].

d. Le plan de déploiement

L'approche à composants SOFA propose un méta-modèle du plan de déploiement inspiré de celui de D&C. Ce méta-modèle permet de décrire l'ensemble des placements à effectuer afin de déployer une application. Le placement des composants se fait sur un nœud centralisé (SOFANode), les composants seront instanciés à distance sur des Docks. La figure 10 présente ce métra-modèle du plan de déploiement SOFA.

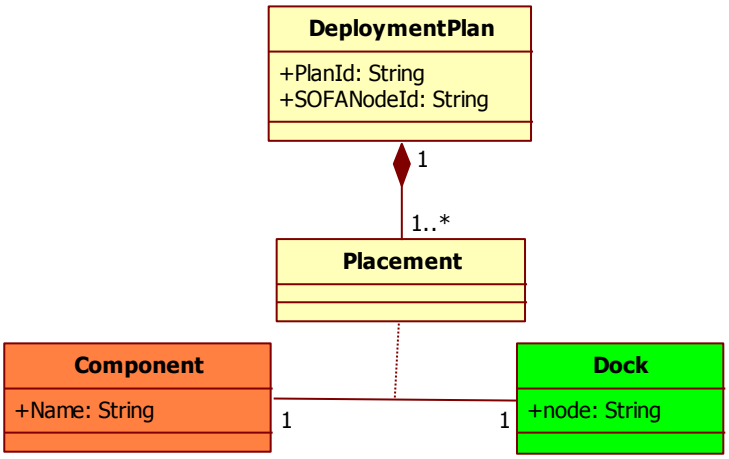


Figure 10 : Le méta-modèle du plan de déploiement SOFA

Le modèle de plan de déploiement présenté ci-dessous décrit le processus de déploiement d'une application SOFA sur un serveur centralisé *sofanodeA* et son processus d'instanciation sur des serveurs distants (*dockA*, *dockB*).

```
<?xml version="1.0" encoding="UTF-8"?>
<depl-plan name="plan1" sofanode="sofanodeA">

   <depl-subc name="article-SOFA-v3.jar" node="dockA"/>
   <depl-subc name="commande-SOFA-v3.jar" node="dockB"/>

</depl-plan>
```

e. Synthèse

Un point négatif de SOFA est qu'il ne propose que du déploiement centralisé de composants logiciels. Le point positif de SOFA est qu'il supporte l'adaptation dynamique des composants.

3.4.3 Le déploiement dans les ADL

a. Le modèle d'application

Il existe une classification et une taxonomie du modèle d'application dans les ADL. Ces modèles d'applications sont [Shaw and Garlan, 1995] :

- Les modèles structurels permettent de décrire les composants, les connecteurs, la configuration architecturale, les interfaces, la sémantique, les contraintes et les propriétés non-fonctionnelles. Aesop, C2, Darwin, Unico, Wright en sont des exemples.
- Les modèles Cadriciels (*Framework*) sont des modèles contraints dans un domaine spécifique (technologie cible, classes spécifiques d'un domaine). EJB, CCM, .NET, Fractal, SOFA en sont des exemples.
- Les modèles dynamiques sont des modèles cadriciels qui décrivent comment reconfigurer et faire évoluer un système. Archetype, Rex, Conic, Rapide en sont des exemples.
- Les modèles à Processus sont des architectures constructives et impératives. Ils ne sont pas hiérarchiques comme on peut le définir dans les modèles structurels. L'architecture applicative se focalise sur la réalisation de tâches conditionnelles qui réalisent le système.
- Les modèles fonctionnels, dans ce modèle l'architecture logicielle est vue comme un ensemble de composants fonctionnels organisés en niveaux. Les composants d'un niveau fournissent des services aux composants du niveau au dessus.

b. Le modèle de domaine

Les modèles d'applications décrits seront déployés sur des modèles de domaine qui peuvent être classés comme suit :

- Architectures n-tiers, ce sont des systèmes hiérarchiques où chaque couche produit un service à la couche au dessous et sert de client pour la couche au dessous. L'architecture client-serveur en est un exemple.
- Grilles de calcul (Grid computing) [Berman et al., 2003] [Foster et al., 2001], se sont des infrastructures virtuelles constituées d'un ensemble de ressources informatiques partagées, distribuées, hétérogènes, délocalisées et autonomes. Cette infrastructure est qualifiée de virtuelle car les relations entre les entités qui la composent n'existent pas sur le plan matériel mais d'un point de vue logique. Elle garantit des qualités de service non triviales, c'est-à-dire qu'elle se distingue des autres infrastructures dans son aptitude à répondre adéquatement à des exigences (accessibilité, disponibilité, fiabilité, …) compte tenu de la puissance de calcul ou de stockage qu'elle peut fournir.
- Grappe de calcul (Cluster computing) [Buyya, 1999] [Baker et al., 1995] est une infrastructure constituée d'un ensemble de ressources informatiques homogènes et localisées, organisées en grappe. Le cluster de calcul désigne des techniques

consistant à regrouper plusieurs ordinateurs indépendants afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur pour augmenter la disponibilité ; faciliter la montée en charge ; permettre une répartition de la charge et faciliter la gestion des ressources (processeur, mémoire vive, disques dur, bande passante réseau).

- Nuage de calcul (Cloud computing) [Ramakrishnan et al., 2010] [Kim et al., 2009], fait référence à l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde entier et liés par un réseau. Les entreprises utilisateurs du *Cloud* ne sont plus propriétaires de leurs serveurs informatiques mais peuvent ainsi accéder de manière évolutive à de nombreux services en ligne sans avoir à gérer l'infrastructure sous-jacente, souvent complexe. Il existe quatre modèles d'infrastructure sur Nuage (*Cloud*) : privé, communautaire et publique.
    - Privé, l'infrastructure de déploiement est exploitée seulement par une organisation.
    - Communautaire, l'infrastructure est partagée entre plusieurs organisations et soutient une communauté spécifique qui partage des préoccupations.
    - Public, l'infrastructure est rendue disponible au grand public ou un grand groupe d'industrie et appartient à une organisation vendant des services de nuage.

c. Les stratégies de déploiement

La notion de stratégies de déploiement n'existe pas dans la majorité des ADL. Certaines stratégies de déploiement sont décrites dans les modèles Cadriciels présentés dans la section 3.2 (EJB, CCM, .NET, SOFA, Fractal).

d. Le plan de déploiement

Nous avons deux visions des ADL, (1) dans le domaine de l'ingénierie des systèmes, un ADL est un langage et un modèle conceptuel utilisé pour décrire et pour représenter un système architectural et (2) dans le domaine de l'ingénierie des logiciels, un ADL est un langage machine utilisé pour décrire et pour représenter une architecture logicielle.

Le plan de déploiement recouvre un concept simple. En effet, un plan de déploiement exprime comment une application peut être distribuée sur un environnement cible d'exécution. En choisissant la première vision, nous en déduisons qu'un ADL peut décrire un plan de déploiement. UML en est un exemple.

e. Synthèse

Les ADL ont une forte puissance d'expression pour la description des architectures matérielles et des architectures logicielles. IL serait intéressant de proposer de nouveaux ADL faisant le mapping entre l'architecture matérielle et l'architecture logicielle, ce travail [Dufrêne and Seinturier, 2008] en est un exemple.

3.4.4 Le déploiement dans UML

a. Le modèle d'application

Le diagramme de composant décrit l'architecture logicielle de l'application comme les paquetages de composants, les fichiers sources, les exécutables et les bibliothèques ainsi que les fichiers de configuration. Dans UML, l'artefact est la représentation physique d'un composant.

b. Le modèle de domaine

Il n'existe pas de diagramme spécifique UML pour représenter uniquement une infrastructure de déploiement. Mais, il existe un diagramme appelé diagramme de déploiement qui est à la fois une représentation du domaine et de l'application. Le diagramme de déploiement permet de représenter les sites de l'environnement cible d'exécution des composants mais ces sites ne sont pas nécessairement du matériel. Certains types de logiciels comme ceux qui fournissent un environnement dans lequel d'autres composants peuvent s'exécuter, sont également des sites. Un ordinateur de bureau, un conteneur J2EE, un serveur web en sont des exemples [Miles and Hamilton, 2006].

c. Les stratégies de déploiement

La notion de stratégies de déploiement n'est pas prise dans UML.

d. Le plan de déploiement

Le diagramme de déploiement permet de décrire l'infrastructure cible (les nœuds), l'application à déployer (les composants) et les dépendances (artefacts). Ce diagramme est utilisable pour le déploiement de petites applications sur de petites infrastructures. Dans de tel cas, le calcul et l'exécution du plan peuvent se faire de façon ad' hoc. Ainsi, l'administrateur système se base sur ce diagramme pour procéder au déploiement. Mais, lorsque le déploiement est à large échelle, il est nécessaire d'automatiser le calcul et l'exécution du plan de déploiement.

e. Synthèse

Le diagramme de déploiement est une vue statique du plan de déploiement et il n'est pas exécutable.

3.5 L'ÉTUDE COMPARATIVE

a. Le déploiement dans les intergiciels

L'avantage des propositions industrielles comme EJB [Dochez, 2009], CCM [OMG, 2006a] et .Net [Troelsen, 2008a, Troelsen, 2008b] est qu'elles permettent de répondre à des besoins spécifiques. Leur désavantage est qu'elles offrent un niveau d'abstraction très bas et chaque activité de déploiement est manuelle. Cela permet de déduire qu'il y a un vrai besoin d'uniformiser le déploiement d'applications dans de tels contextes. Ces propositions industrielles ne supportent pas la description de l'infrastructure de déploiement. Ils fournissent peu de sémantique pour décrire les applications, par exemple, les besoins d'une application peuvent être une version spécifique d'un logiciel, et une taille mémoire supérieure à 10 GO. Aucune de ces contraintes ne sera vérifiée lors de l'installation, cela correspond à des recopies d'assemblages de composants. Le plan de déploiement exprime comment une application doit être déployée mais il est décrit de différentes manières (syntaxe différente) pour chaque intergiciel.

b. Le déploiement dans la spécification D&C de l'Omg

Les industriels ont senti la nécessité de joindre leurs efforts. Ils ont proposé une spécification qui capitalise les expériences de l'industrie dans le déploiement (l'approche de l'OMG). Cette spécification a inspiré beaucoup d'académiques. La spécification OMG D&C (*Deployment and Configuration*) [OMG, 2006b] est basée sur l'utilisation de modèles et de leurs transformations. Cette spécification standardise beaucoup d'aspects du déploiement pour des systèmes distribués à base de composants, y compris l'assemblage de composants, la configuration des assemblages et la gestion des ressources de l'environnement cible d'exécution. Ces aspects sont traités via un modèle de données et un modèle d'exécution. Le modèle de données peut être utilisé pour définir/générer des schémas XML pour stocker et échanger les métadonnées qui décrivent les assemblages de composants, leur configuration et les caractéristiques du déploiement. Le modèle d'exécution définit des processus qui traitent les métadonnées décrites dans le modèle de données pendant le déploiement du système. L'OMG est aussi à la base de l'approche à composants CCM (Corba Component Model). De ce fait, la seule implémentation disponible de D&C est CCM [Deng et al., 2005]. Le vrai problème qui se pose est que l'OMG a une vision Corba des composants. De ce fait, l'extension vers d'autres approches à composants industriels comme EJB, .NET est difficilement envisageable.

c. Le déploiement à haut niveau d'abstraction

Dans les modèles à composants actuels comme OSGI [Alliance, 2005], les Services Web [Gustavo et al., 2004], SOFA [Bures et al., 2006] et UML 2.0 [OMG, 2007], les composants sont définis sous forme d'unités architecturales [Kaur and Singh, 2009]. Les ADL [Medvidovic and Taylor, 2000] comme Acme, AADL, Darwin et Wright permettent de modéliser les composants, de modéliser les connecteurs et de

modéliser les différentes configurations d'architecture. Mais le processus de déploiement reste non spécifié.

d. Les tableaux de comparaison

Nous avons retenu quatre critères principaux pour comparer des environnements de déploiement : l'application, le domaine, les stratégies et le plan de déploiement.

L'application, un méta-modèle d'application doit exister pour tout environnement de déploiement. Le modèle d'application sera conforme à ce méta-modèle d'application et il permettra de décrire :

- L'architecture logicielle c'est-à-dire l'ensemble des composants qui forment l'application.
- Les contraintes logicielles c'est-à-dire l'ensemble des dépendances vers des ressources logicielles.
- Les contraintes matérielles c'est-à-dire l'ensemble des dépendances vers des ressources matérielles.

Le domaine, un méta-modèle de domaine doit exister pour tout environnement de déploiement. Le modèle de domaine sera conforme à ce méta-modèle de domaine et il permettra de décrire :

- L'architecture matérielle c'est-à-dire l'ensemble des sites interconnectés qui forment le domaine.
- Les ressources logicielles et matérielles disponibles.

Les stratégies, un méta-modèle de stratégies doit exister pour tout environnement de déploiement. Le modèle de stratégies sera conforme à ce méta-modèle de stratégies et il permettra de décrire :

- Les stratégies techniques qui expriment quelle influence a aussi bien l'architecture matérielle que logicielle sur le cycle de vie logicielle.
- Les stratégies d'entreprise qui sont propres à chaque organisation et qui permettent aux entreprises de personnaliser le déploiement.

Le plan de déploiement, un méta-modèle de plan de déploiement doit exister pour tout environnement de déploiement. Le modèle de plan de déploiement sera conforme à ce méta-modèle de déploiement et il doit :

- être obtenu à partir d'un processus de planification qui prend en entrée le modèle d'application, le modèle de domaine, le modèle de stratégies afin de produire ce modèle de plan de déploiement.
- contenir toutes les informations nécessaires pour mener à bien toutes les activités de déploiement.
- être fourni sous un format spécifique.
- être exécutable par un environnement cible.

Tableau 1 : Tableau de comparaison (Application et Domaine)

Approches	Méta-modèle d'application				Méta-modèle de domaine			
	Architecture logicielle	Contraintes logicielles	Contraintes matérielles	Format du descripteur	Architecture matérielle	Ressources logicielles	Ressources matérielles	Format du descripteur
EJB	*			Conforme au DTD ejb-jar				
CCM	*	*	*	Conforme aux DTD SoftwarePackageDescriptor.dtd CORBAComponentDescriptor.dtd				
Canevas .Net	*	* (seulement les dépendances entre assemblages)		Manifest MSI				
D&C	*	*	*	Meta-Modèles UML à partir desquels peut être générés des fichiers xml et IDL ComponentDataModel ComponentManagementModel	*	*	*	Meta-Modèles UML à partir desquels peut être générés des fichiers xml et IDL TargetDataModel TargetManagement Model
Software Dock	*	*	*	Conforma au DTD DSD	*	*	*	Fieldock Releasedock (agent)
Orya	*	*	*	Modèle de produit (diagramme de classe UML)	*	*	*	Modèle de site (diagramme de classe UML)
Fractal	*	*	*	Fractal ADL (xml)				
SOFA	*			SOFA component meta-model	* Docks (nœuds distants qui peuvent instancier les composants)			Sofanode (nœud centralisé pour le placement des composants)
UML	*	*	*	Diagramme de composant	*	*	*	Diagramme de déploiement

* (activité prise en compte)

Tableau 2 : Tableau de comparaison (Stratégies et plan de déploiement)

Approches	Méta-modèle de stratégies				Méta-modèle de plan de déploiement			
	Techniques	Entreprise	Flexibles /Figées	Langage de spécification de stratégies	Processus de planification supporté	Plan de déploiement complet	Plan de déploiement exécutable	Format du plan de déploiement
EJB	*		Figées					Script
CCM	*		Figées	SoftwarePackageDescriptor.dtd CORBAComponentDescriptor.dtd CORBAassemblyDescriptor.dtd				Script
Canevas .Net	*		Figées	*(seulement pour les mise-à-jour)				
D&C					*	*	*	Meta-modèle UML au niveau PIM (Schéma xml pour CCM/Dance)
Software Dock	*(configura-tion)		Figées		*	*		Plan de déploiement enfouis dans l'outil (code)
Orya		* (peu de sémantique)	Flexibles	Modèle de stratégies	*	*		Plan de déploiement enfouis dans l'outil (code)
Fractal	*		Figées					
SOFA	*		Figées	* (seulement pour l'adaptation dynamique DCUP)		*	*	Document XML
UML								Diagramme de déploiement

* (activité prise en compte)

4

LE CADRE CONCEPTUEL DE NOTRE APPROCHE : ARCHITECTURE ET META-MODELES

4.1 L'ARCHITECTURE

Au regard du bilan obtenu à partir de l'état de l'art de la pratique, nous pensons qu'une bonne solution de déploiement automatisé pour les systèmes à composants doit [Dibo and Belkhatir, 2010a] :

- couvrir toutes les activités de déploiement,
- être indépendante des technologies,
- être indépendante de toute philosophie d'approche à composants,
- offrir un moteur de déploiement distribué,
- proposer un langage de spécification de stratégies afin de rendre le déploiement flexible et de supporter des stratégies existantes dans les environnements de déploiement.

L'analyse d'un système de déploiement fait ressortir des activités indépendantes des technologies et que l'on pourrait factoriser comme :

- la modélisation de l'application à déployer,
- la modélisation de l'environnement d'exécution des composants
- le calcul du plan de déploiement.

De ce fait, nous proposons une architecture de déploiement [Dibo and Belkhatir, 2010c] basée sur une approche MDA (Model-Driven Architecture) [OMG, 2005b] avec l'utilisation de modèles, de méta-modèles et de leur transformation (l'approche MDA est décrite dans la section 4.2). Adopter une approche MDA va permettre d'offrir un cadre unifié basé sur des activités de déploiement utilisant des descripteurs génériques qui pourront être par la suite personnalisé pour des plateformes spécifiques.

L'étude du déploiement dans les pratiques d'entreprise nous a permis de comprendre que le déploiement doit être flexible suivant les besoins de l'entreprise et suivant les spécificités techniques de l'application. Ainsi, nous proposons un quatrième métamodèle relatif aux stratégies de déploiement qui vient en complément des trois metamodèles communs.

La figure 11 schématise ce processus de déploiement constitué de six activités principales qui sont :

- La modélisation de l'application (*application modeling*) qui permet de décrire l'application à déployer, c'est-à-dire de spécifier l'ensemble des composants qui la constituent et les contraintes en ressources de ces composants.
- La modélisation du domaine (*domain modeling*) qui permet de décrire l'environnement de déploiement, c'est-à-dire de spécifier l'ensemble des sites qui le constituent et les ressources disponibles.

– La modélisation des stratégies de déploiement (*strategy modeling*) qui permet de décrire les politiques à mettre en œuvre pour rendre flexible le calcul du plan selon des besoins spécifiques.

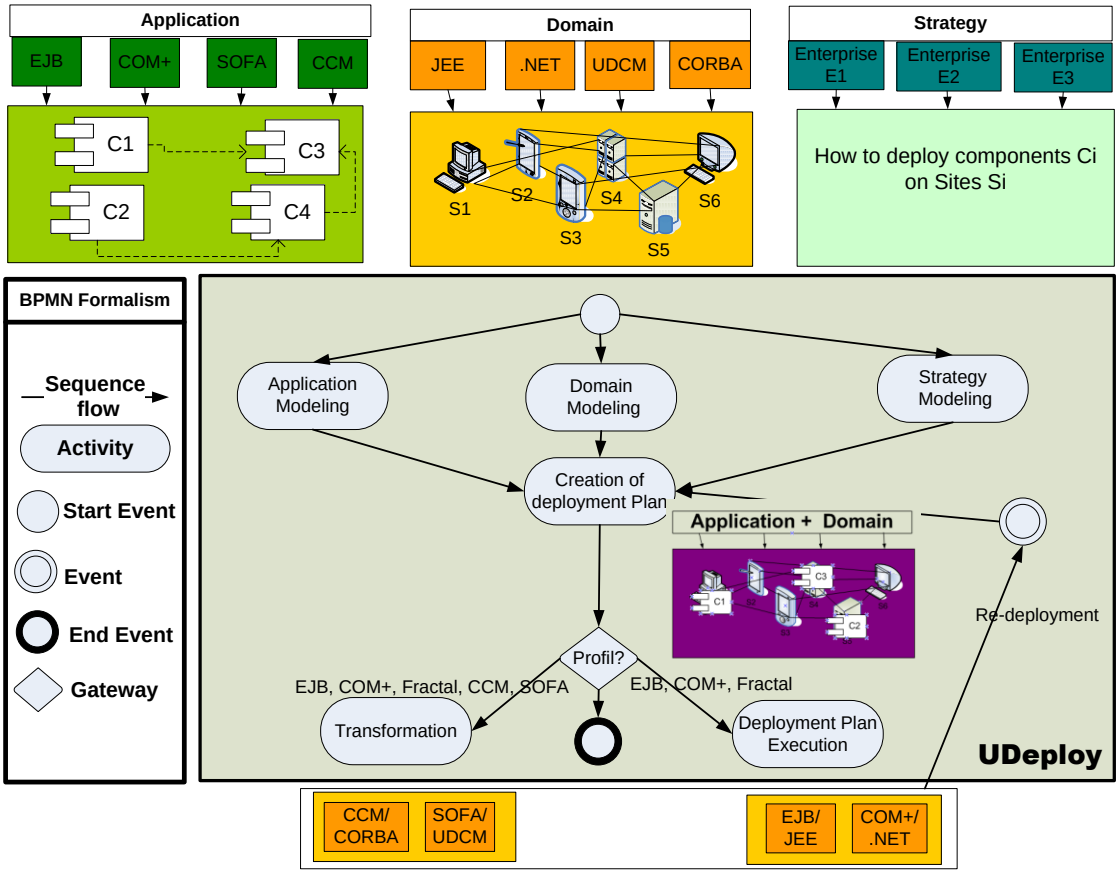


Figure 11 : L'architecture de l'environnement UDeploy (Unified Deployment architecture)

– Le calcul du plan de déploiement (*creation of deployment plan*) qui à partir d'un modèle d'application, d'un modèle de domaine et d'un modèle de stratégies produit un plan de déploiement.

– La transformation couvre deux activités principales qui sont

  - La personnalisation du plan de déploiement. Le plan de déploiement produit à la fin de l'activité de calcul du plan est au niveau PIM (*Platform independent model*) c'est-à-dire indépendante de toute technologie. Ce plan est vu comme un ensemble de placements. Ce plan générique doit être personnalisé vers un ou des plans au niveau PSM (*Plateform specific model*) c'est-à-dire spécifiques à des technologies pour qu'ils puissent être exécutés par les intergiciels cibles. Le plan de déploiement répond à la question « où déployer ? ».

  - La génération du descripteur de déploiement. Le descripteur de déploiement est construit à partir d'informations contenues dans le modèle d'application et d'autres informations produites par le *deployeur* sur les propriétés non-fonctionnelles de l'application. Le descripteur de

déploiement répond à la question « comment le conteneur doit gérer les composants à déployer ».

– L'exécution du plan de déploiement (*deployment plan execution*). Certains intergiciels n'offrent pas de support pour l'exécution du plan de déploiement. Dans ce cas, le plan générique sera traduit dans une description propre aux intergiciels (script). Cette description sera exécutée par notre outil de déploiement en invoquant les méthodes de l'intergiciel cible.

Dans ce qui suit, nous décrivons les différents méta-modèles (application, domaine, stratégies, plan de déploiement et descripteur de déploiement) utilisés dans les différents processus qui sont la modélisation de l'application, la modélisation du domaine, la modélisation des stratégies de déploiement, la création du plan de déploiement, la personnalisation et l'exécution du plan de déploiement.

4.2 L'APPROCHE MDA

4.2.1 Les concepts clés

L'approche MDA [OMG, 2005b] a été proposée par l'OMG pour répondre aux problématiques que posent la multiplicité des systèmes informatiques, des langages et des technologies. L'idée principale de l'approche MDA est la séparation des préoccupations techniques et métiers [Gerber et al., 2002]. Les concepts clés inhérents à l'approche MDA sont :

- Le **PIM** (*Plateform Independant Model*), ces modèles sont indépendants des plateformes technologiques comme par exemple EJB, CCM, COM+ et offrent un haut niveau d'abstraction.
- Le **PSM** (*Plateform specific Model*), ces modèles sont dépendants des plateformes technologiques et correspondent au code exécutable.
- La **transformation**, le passage PIM vers PSM ou PSM vers PIM se fait par des transformations de modèles. Une transformation de modèle est définie à partir d'un ensemble de règles. Ces règles peuvent être décrites en utilisant un outil de transformation de type QVT (Query View Transformation) [OMG, 2005a], ou en implémentant sa propre machine de transformation. Il existe plusieurs outils et langages de transformations comme les *QVT-core* (MTF [IBM, 2010]), les *QVT-relations* (medini QVT [mediniQVT, 2010]) et les *QVT-like (*ATL [Jouault et al., 2006] [Jouault et al., 2008], Tefkat [Lawley and Steel, 2005] et VIATRA [Varró and Balogh, 2007]).

4.2.2 Les avantages de l'approche MDA

Les avantages principaux de l'approche MDA sont la productivité et la portabilité [Kleppe et al., 2003]. La **productivité** parce que les développeurs pourront ainsi se concentrer sur le développement des modèles PIM. Ils travailleront à un niveau où les détails techniques ne seront pas spécifiés. Ces détails techniques seront ajoutés au niveau PSM au moment de la transformation. Cela améliore la productivité de deux façons. En premier lieu, les développeurs de PIM omettent les détails spécifiques. En deuxième lieu, plusieurs PSM peuvent être obtenus pour différentes plateformes avec moins d'effort. La **portabilité** parce qu'un PIM peut être automatiquement transformé en plusieurs PSM pour différentes plates-formes. Ainsi, tout ce qui est spécifié au niveau PIM reste donc portable. Il suffirait pour cela que le code à générer soit conforme à la technologie d'une plate-forme cible d'exécution.

4.2.3 L'approche MDA et le déploiement

Les outils de déploiement classiques intégrés dans les intergiciels redéveloppent de manière spécifique les mécanismes et les processus de déploiement. Ces outils peuvent être vus comme étant au niveau PSM. Donc appliquer MDA au déploiement reviendrait à définir des méta-modèles abstraits de déploiement qui seront au niveau PIM et qui pourront être personnalisés pour différentes plateformes.

4.3 LES DIFFERENTS META-MODELES CONSTITUANT NOTRE SYSTEME DE DEPLOIEMENT

4.3.1 Le méta-modèle d'application de UDeploy

La majorité des approches à base de composants logiciels et des environnements de déploiement offrent un méta-modèle d'application. Ce méta-modèle décrit l'architecture logicielle, les contraintes logicielles et les contraintes matérielles. Le tableau 1 dresse un bilan des différents méta-modèles d'application.

- Les méta-modèles d'application EJB et SOFA permettent de décrire l'architecture logicielle.
- Le méta-modèle d'application du canevas .NET permet de décrire l'architecture logicielle et les contraintes logicielles partiellement (seulement les dépendances entre assemblages).
- Les méta-modèles d'application CCM, D&C, Software Dock, Orya et Fractal permettent de décrire l'architecture logicielle, les contraintes logicielles et les contraintes matérielles.

Les méta-modèles d'application étant nombreux, nous proposons un méta-modèle d'application pour la description des architectures logicielles (figure 12). Il se veut indépendant de toute technologie et de toute plate-forme particulière. Il permet d'uniformiser la description des applications et permet que les approches à composants qui n'ont pas une sémantique forte pour la description des architectures logicielles puissent être soutenues. Le modèle d'application d'un logiciel à déployer est un descripteur dont la syntaxe est conforme à notre méta-modèle et contiendra les informations suivantes :

- Le producteur de l'application « *application producer* ».
- La liste des composants qui la constituent (déploiement big-bang ou immédiat) ou la liste de quelques composants qui constituent l'application (déploiement progressif ou reprise après panne comme par exemple le redéploiement des composants installés sur un nœud en panne) « *component* ».
- La description de chaque implémentation, c'est-à-dire le standard de spécification qui peut être un modèle à composants exécutables comme EJB, CCM et NET ou un modèle à composants hiérarchiques pas nécessairement exécutables comme Fractal, Sofa, Darwin et Kaola « *implementation* ». La description du code de l'implémentation, c'est-à dire le nom de l'archive qui contient le code et la localisation de l'implémentation dans la base de composants « *repository* ».
- Le langage de programmation des implémentations (Java, C++) et le langage humain des implémentations (de, en, fr, es, pt, it). La connaissance du langage humain pour les composants de type présentation est intéressante car cela permet de rendre flexible le déploiement.

– Les dépendances fortes (assertions) des implémentations (*requirement « dependencetype » « dependencyassert »*). Elles ne peuvent pas être résolues pendant le processus de déploiement. Elles s'expriment en imposant des valeurs d'attributs pour le compilateur, l'OS, le processeur, le moteur d'exécution « *runtime* » et l'intergiciel « *middleware* ».

– Les dépendances faibles ou dépendances à l'installation des implémentations (*requirement « dependencetype » « dependencyinstall »*). Elles peuvent être résolues pendant le processus de déploiement. Elles expriment comment installer et où chercher le compilateur, le moteur d'exécution, l'intergiciel et les fichiers (librairies et exécutables) indispensables au bon fonctionnement de l'implémentation.

– La cardinalité de chaque composant pour une application spécifique (*cardinality « unique » « multiple »*) exprimant la possibilité de déployer sur un ou plusieurs sites.

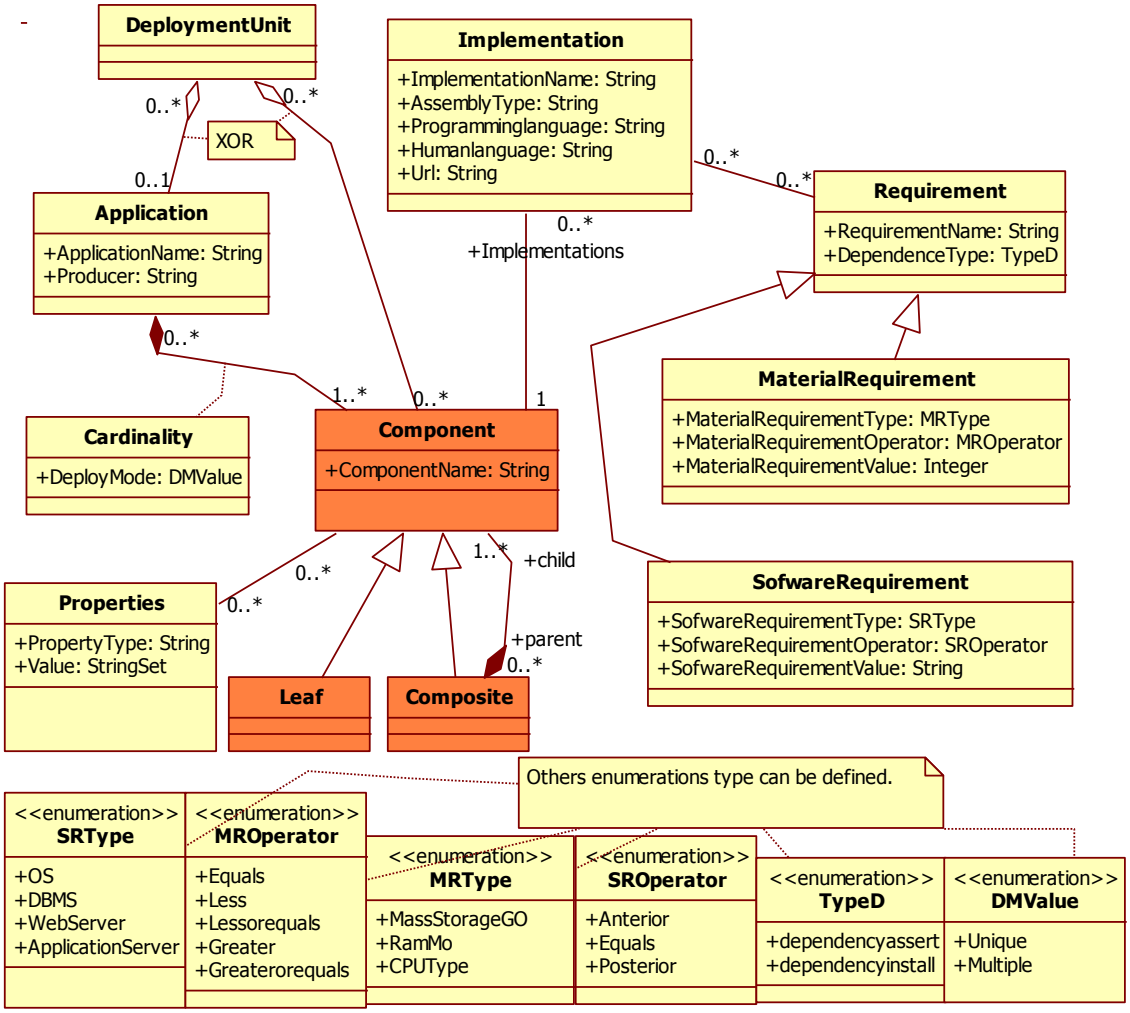


Figure 12 : Le métamodèle d'application UDeploy

Le modèle d'application UDeploy ci-dessous décrit une application composée de deux composants *article* et *localisation*.

Le composant *article* est de type *EJB entity*. Il a comme contraintes logicielles un moteur d'exécution Java, un serveur de base de données et un serveur d'application et comme contrainte matérielle une mémoire de stockage égale à 450 GO.

Le composant *localisation* est de type *CCM session* et a comme contrainte logicielle un moteur d'exécution Java.

– Les caractéristiques de base de l'application (nom et producteur)

```
<applicationName>scorp</applicationName>
<producer>abylan</producer>
```

– L'ensemble des composants qui forment l'application (nom du composant, nom de l'implémentation, type de composant, langage de programmation et url)

```
<components>
   <component>
      <componentName>article</componentName >
      <implementationName>article-ejb-v3.jar</implementationName >
      <assemblyType>EJB entity</ assemblyType>
      <programmingLanguage>Java</programmingLanguage>
      <url>D:\repository\applications\</url>
      <cardinality>Unique</cardinality>
   </component>

   <component>
      <componentName>localisation</componentName >
      <implementationName>localisation-ccm-v3.jar</implementationName>
      <assemblyType>CCM Session</assemblyType>
      <programmingLanguage>Java</programmingLanguage>
      <url>D:\repository\applications\</url>
      <cardinality>Multiple</cardinality>
   </component>
</components>
```

– Les contraintes logicielles (nom de la contrainte, type de dépendance, type de contrainte, opérateur et valeur)

```
<softwareRequirements>

   <softwareRequirement>
      <requirementName>SR1</requirementName >
      <dependenceType>dependencyinstall</dependenceType >
      <srType>runtime</srType>
      <srOperator>Equals</srOperator>
      <srValue>JRE1.4.2-02</srValue>
   </softwareRequirement>

   <softwareRequirement>
      <requirementName>SR2</requirementName >
      <dependenceType>dependencyinstall</dependenceType >
      <srType>J2EE server</srType>
      <srOperator>Equals</srOperator>
      <srValue>JBOSS4.2.3.GA</srValue>
   </softwareRequirement>

   <softwareRequirement>
      <requirementName>SR3</requirementName >
      <dependenceType>dependencyinstall</dependenceType >
      <srType>Data server</srType>
      <srOperator>Equals</srOperator>
      <srValue>Oracle8.1.5</srValue>
   </softwareRequirement>

</softwareRequirements>
```

– Les contraintes matérielles (nom de la contrainte, type de dépendance, type de contrainte, opérateur et valeur)

```
<materialRequirements>
   <materialRequirement>
      <requirementName>MR1</requirementName >
      <dependenceType>dependencyassert</dependenceType >
      <mrType>MassStorageGO</mrType>
      <mrOperator>Equals</mrOperator>
      <mrValue>450</mrValue>
   </materialRequirement>
</materialRequirements>
```

– Les contraintes logicielles et matérielles associées à chaque composant (le nom de l'implémentation et la liste des contraintes qui lui sont associées)

```
<constraints>
   <constraint>
      <implementationName>article-ejb-v3.jar</implementationName>
      <requirementNames>
              <requirementName>SR1</requirementName>
              <requirementName>SR2</requirementName>
              <requirementName>SR3</requirementName>
              <requirementName>MR1</requirementName>
      </requirementNames>
   </constraint>

   <constraint>
      <implementationName >locasisation-ccm-v3.jar</implementationName>
      <requirementNames>
              <requirementName>SR1</requirementName>
              …
      </requirementNames>
   </constraint>
</constraints>
```

Le méta-modèle UDeploy permet de décrire l'architecture logicielle, les contraintes logicielles et les contraintes matérielles d'une application. Par contre, il ne permet pas de décrire les propriétés non fonctionnelles comme la transaction, la persistance et la sécurité.

Nous avons retardé la définition des propriétés non-fonctionnelles. En effet, nous pensons que les propriétés non fonctionnelles doivent être définies au niveau du descripteur de déploiement et non au niveau du modèle d'application. Cela permet, le déploiement d'une même application de différentes manières (propriétés non-fonctionnelles flexibles).

4.3.2 Le méta-modèle de domaine de UDeploy

Nous proposons un méta-modèle de domaine schématisé par la figure 13 pour la description des architectures matérielles, des ressources logicielles et des ressources matérielles.

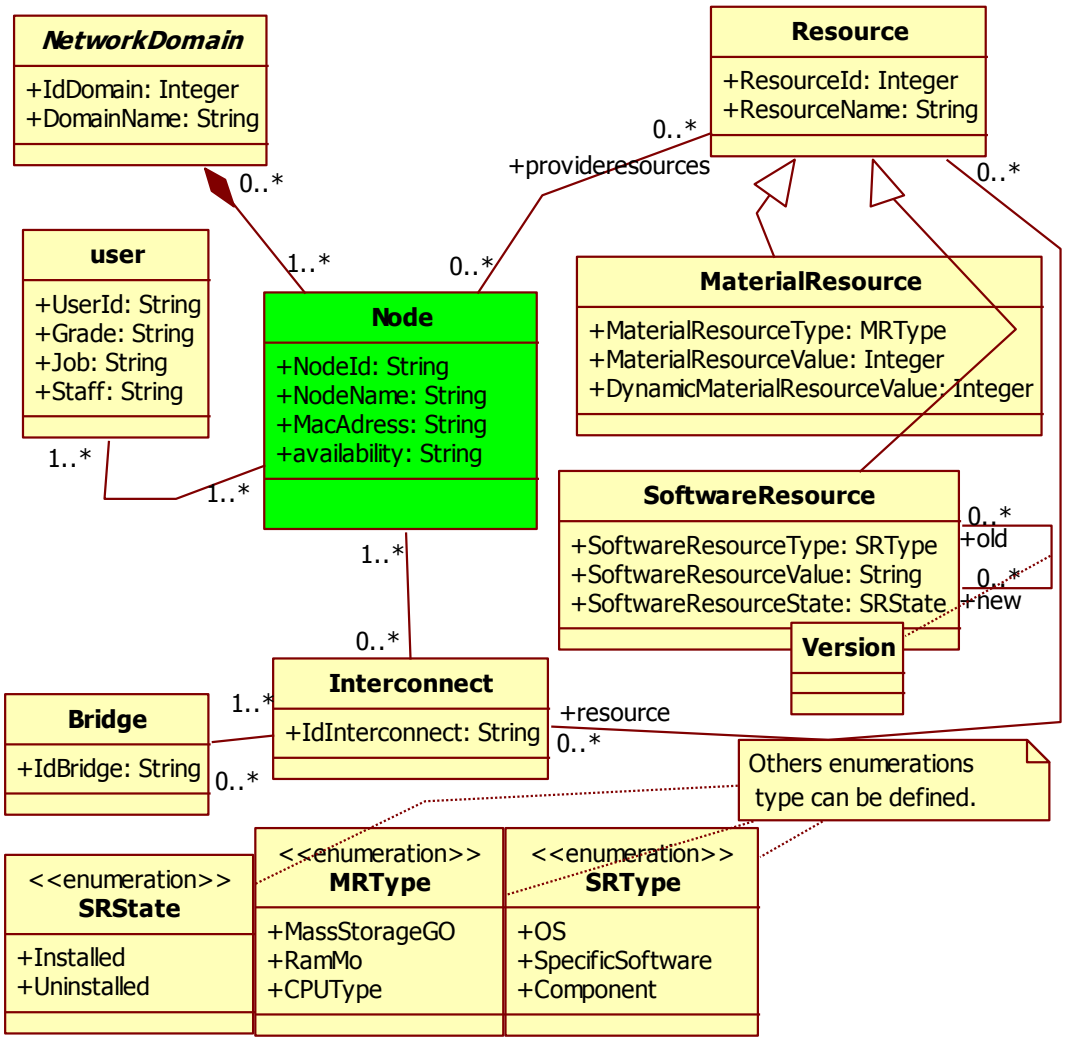


Figure 13 : Le métamodèle de domaine UDeploy

Le modèle de domaine d'un environnement d'exécution est un descripteur dont la syntaxe est conforme à notre méta-modèle. Le modèle de domaine est mis à jour à chaque fois qu'un événement se passe sur le domaine. Il contiendra les informations suivantes :

– Le nom du domaine.

– La liste des nœuds qui constituent le domaine selon leur disponibilité (attribut *availability*)

– La description des ressources logicielles pour chaque nœud qui peuvent être le compilateur, l'OS, le processeur, le moteur d'exécution (*runtime)*, l'intergiciel (*middleware*) et les fichiers (librairies et exécutables).

– La description des ressources matérielles pour chaque nœud qui peuvent être la mémoire vive, la mémoire masse, la vitesse du processeur.

– La description des liens réseaux entre les nœuds.

– La description des ressources sur les liens réseaux (latence, débit, taux de perte, gigue).

– La liste des composants installés sur chaque nœud lors de précédentes activités de déploiement.

– La localisation des nœuds.

– Les utilisateurs des sites avec leur métier (comptable, agent de saisie), leur grade (responsable, chef de projet) et leur équipe (RH, SAV, Marketing). Les machines de type serveur sont liées à l'administrateur système.

Le modèle de domaine UDeploy présenté ci-dessous décrit un domaine composé de trois sites *H1*, *H2* et *H3*. Le site *H1* est disponible et a comme ressources logicielles un moteur d'exécution Java, un serveur de base de données et un serveur d'application et comme ressources matérielles une mémoire de stockage égale à 900 Go et une mémoire de stockage dynamique égale à 850 euros. Cela veut dire qu'une réservation de 50 Go (900 Go-850 Go) de mémoire masse a été faite pour ce nœud. Le plan de déploiement qui requiert ces 50 Go n'a pas encore été exécuté. Le site *H2 est disponible et* a comme ressource logicielle un moteur d'exécution Java. Le site *H3* est indisponible au moment où le modèle de domaine a été généré.

– Le domaine (nom du domaine)

```
<domaineName>adele</applicationName>
```

– Les nœuds du domaine (nom du nœud, adresse ip, disponibilité)

```
<nodes>

   <node>
      <nodeName>H1</nodeName >
      <nodeId>145.25.35.145</nodeId>
      <availability>True</availability>
   </node>

   <node>
      <nodeName>H2</nodeName >
      <nodeId>145.25.35.146</nodeId>
      <availability>True</availability>
   </node>

   <node>
      <nodeName>H3</nodeName >
      <nodeId>145.25.35.147</nodeId>
      <availability>False</availability>
   </node>

</nodes>
```

– Les ressources logicielles (nom de la ressource logicielle, type de ressource, valeur, état de la ressource)

```
<softwareResources>

   <softwareResource>
      <resourceName>SRE1</resourceName>
      <sresourceType>runtime</sresourceType>
      <sresourceValue>JRE1.4.2-02</sresourceValue>
      <sresourceState>Installed</srOperator>
   </softwareResource>

   <softwareResource>
      <resourceName>SRE2</resourceName>
      <sresourceType>JEE Server</sresourceType>
      <sresourceValue>JBOSS4.2.3.GA</sresourceValue>
      <sresourceState>Installed</srOperator>
   </softwareResource>

   <softwareResource>
      <resourceName>SRE3</resourceName>
      <sresourceType>Data server</sresourceType>
      <sresourceValue>Oracle8.1.5</sresourceValue>
      <sresourceState>Installed</srOperator>
   </softwareResource>

</softwareResources>
```

– Les ressources matérielles (nom de la ressource matérielle, type de ressource, valeur, état de la ressource)

```
<materialResources>

   <materialResource>
      <resourceName>MRE1</resourceName >
      <mresourceType>MassStorageGO</mresourceType>
      <mresourceValue>900</mresourceValue>
      <dynamicMResourceValue>850</dynamicMResourceValue>
   <materialResource>

</materialResources>
```

– Les ressources logicielles et matérielles offertes par chaque nœud (le nom du nœud et la liste des ressources offertes)

```
<resources>

   <resource>
      <nodeId>H1</nodeId>
      <resourceNames>
               <resourceName>SRE1</resourceName>
               <resourceName>SRE2</resourceName>
               <resourceName>SRE3</resourceName>
               <resourceName>MRE1</resourceName>
      </resourceNames>
   </resource>

   <resource>
      <nodeId>H2</nodeId>
      <resourceNames>
               <resourceName>SRE1</resourceName>
      </resourceNames>
   </resource>
```

```
</resources>
```

4.3.3 Le méta-modèle de stratégies de déploiement de UDeploy

L'étude des approches à composants et des environnements de déploiement nous montre que généralement les stratégies de déploiement sont figées et spécifiques aux technologies.

Nous proposons un méta-modèle de stratégies représenté par la figure 14 qui guident la création du plan de déploiement. Une bonne stratégie de déploiement doit permettre d'exprimer les choix techniques et les politiques d'entreprise :

– Les choix techniques expriment quelle influence a aussi bien l'architecture matérielle que logicielle sur le cycle de vie logicielle.

– Les politiques d'entreprise sont propres à chaque organisation, elles permettent aux entreprises de personnaliser le déploiement.

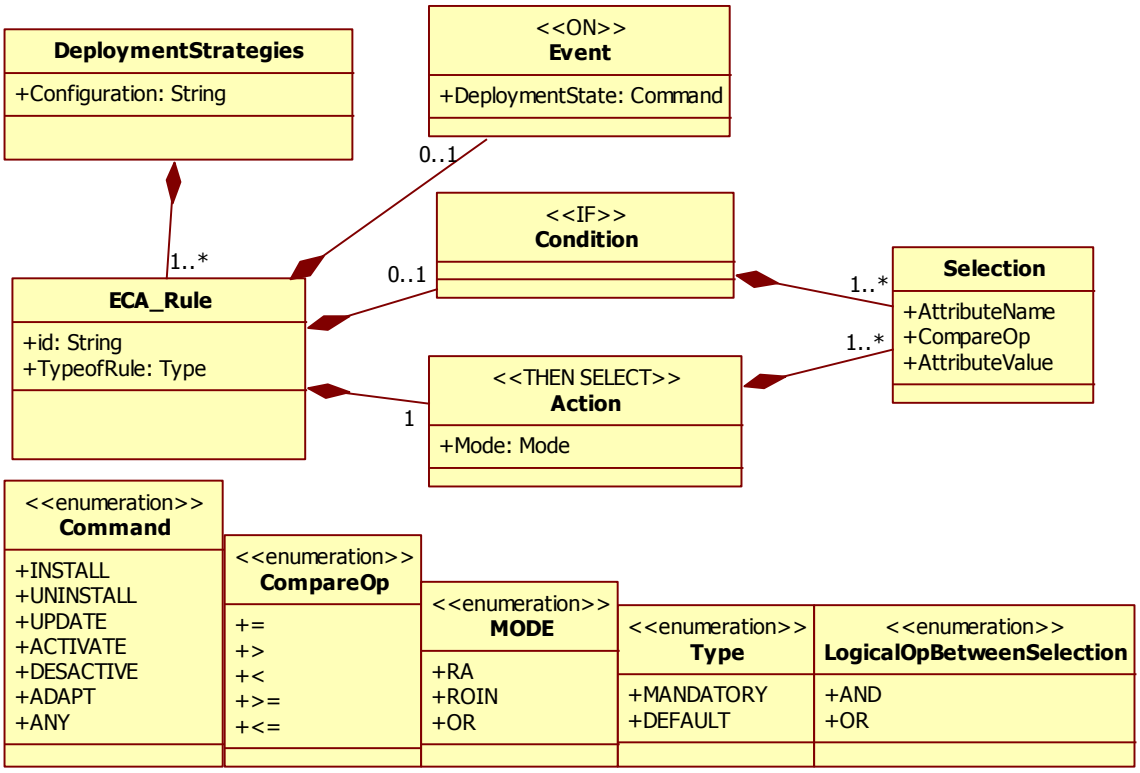


Figure 14 : Le méta-modèle de stratégies UDeploy

Un modèle de stratégies de déploiement est défini suivant des règles ECA [McCarthy and Dayal, 1989] [Dittrich et al., 1995] [Papamarkos et al., 2003]. Il contient une ou plusieurs règles ECA.

on Event if Condition then Action

– Nous avons défini deux types de règles ECA (impératives ou par défaut). Ces règles s'appliquent à l'association du couple composants-sites. Les résultats obtenus doivent satisfaire les contraintes définies par la règle.

- Règles impératives (*Mandatory*) : les composants spécifiés doivent être déployés sur les sites spécifiés.

```
<eca-rule>
   <ruleId>r1</ruleId>
   <typeofRule>Mandatory</typeofRule>
   ON
   <event>
   …
   </event>
   IF
   <condition>
   …
   </condition>
   THEN SELECT
   <action>
   …
   </action>
</eca-rule>
```

- Règles par défaut (*Default)* : les composants et les sites spécifiés par leurs attributs s'appliquent si ces composants et sites existent, sinon la règle n'a aucun effet. Elles ne sont utilisées que par défaut, et si elles ne sont pas incompatibles avec les règles impératives.

```
<eca-rule>
   <ruleId>r2</ruleId>
   <typeofRule>Default</typeofRule>
   ON
   <event>
   …
   </event>
   IF
   <condition>
   …
   </condition>
   THEN SELECT
   <action>
   …
   </action>
</eca-rule>
```

– Un événement spécifie une commande qui invoque le déclenchement d'une règle. Les différentes commandes disponibles sont *install*, *uninstall*, *update,activate*, *desactivate*, *adapt* et *any*.

```
<event>
   <deploymentState>INSTALL</deploymentState>
</event>
```

– Une condition est un test logique sera la cause de l'action à mettre en œuvre. L'action associée à la condition ci-dessous ne portera que sur les composants *EJB entity*, *session* et *message-driven*.

```
<condition>
   <selections>
      <selection>
         <attributeName>component.implementation.assemblyType</attributeName>
         <compareOp>=</compareOp>
         <attributeValue>EJB entity</attributeValue>
      </selection>
      OR
      <selection>
         <attributeName>component.implementation.assemblyType</attributeName>
         <compareOp>=</compareOp>
```

```
            <attributeValue>EJB session</attributeValue>
         </selection>
         OR
         <selection>
            <attributeName>component.implementation.assemblyType</attributeName>
            <compareOp>=</compareOp>
            <attributeValue>EJB message-driven</attributeValue>
         </selection>
      </selections>
   </condition>
```

– Action est la sélection de propriétés spécifiques quand une condition est satisfaite. L'action ci-dessous exprime que pour tout composant *EJB entity*, *session* et *message-driven* le nœud de placement du composant doit fournir un serveur d'application J2EE.

```
   <action>
      <mode>RA</mode>
      <selections>
         <selection>
            <attributeName>
               node.provideResources.softwareResource.softwareResourceType
            </attributeName>
            <compareOp>=</compareOp>
            <attributeValue>J2EE SERVER</attributeValue>
         </selection>
   </action>
```

– Une sélection (*AttributeName*, *CompareOp*, *AttributeValue*) permet de spécifier les propriétés définies dans le modèle d'application pour la partie composant et dans le modèle de domaine pour la partie site. Une action ou une condition est formée de une ou de plusieurs sélections avec des opérateurs booléens entre les sélections (*AND*, *OR*).

```
         <selection>
            <attributeName>component.implementation.assemblyType</attributeName>
            <compareOp>=</compareOp>
            <attributeValue>EJB message-driven</attributeValue>
         </selection>
```

– Pour la partie mode de déploiement nous nous appuyons sur le travail développé par [PDC01] en fonction de la compatibilité des versions de composants :

- *Replace Always (RA)* : toujours remplacer.
- *Replace Only If Newer (ROIN)* : remplacer si seulement le composant est plus récent.
- *Never Replace (NR)* : ne pas remplacer si le composant est déjà déployé.

```
   <action>
      <mode>RA</mode>
      …
   </action>
```

4.3.4 Le méta-modèle du plan de déploiement de UDeploy

Nous proposons un méta-modèle du plan de déploiement schématisé par la figure 15 qui permet de décrire l'ensemble des placements à effectuer pour déployer une application A sur un domaine D. Nous avons fait le choix de limiter la notion de plan de déploiement au concept de placement sans faire intervenir l'ordonnancement.

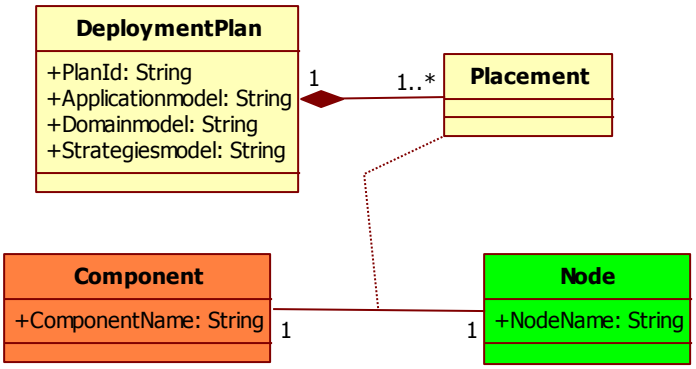


Figure 15 : Le méta-modèle du plan de déploiement UDeploy

Le modèle de plan de déploiement ci-dessous décrit l'ensemble des placements qu'il faut effectuer afin de déployer l'application *scorp*, sur le domaine *adele* en appliquant le modèle de stratégies *modelstrategies1*. Les placements sont : le composant article sur le nœud H1 et le composant localisation sur le nœud H2.

```
<deploymentplanmodel>

   <planId>plan1</planId>
   <applicationModel>scorp</applicationModel>
   <domainModel>adele</domainModel>
   <strategiesModel>modelstrategies1</strategiesModel>

   <placements>

      <placement>
         <componentName>article</componentName >
         <nodeName>H1</nodeName>
      </placement>

      <placement>
         <componentName>localisation</componentName >
         <nodeName>H2</nodeName>
      </placement>

   </placements>

</deploymentplanmodel>
```

Nous verrons par la suite l'algorithme de planification qui à partir d'un modèle d'application, d'un modèle de domaine et d'un modèle de stratégies permet d'obtenir un modèle de plan de déploiement.

4.3.5 Le metamodèle du descripteur de déploiement de UDeploy

Le modèle de descripteur de déploiement décrit l'architecture logicielle c'est-à-dire l'ensemble des composants qui constituent une application ainsi que les propriétés non fonctionnelles de l'application telles que la sécurité, les transactions, la tolérance aux pannes et la persistance ainsi que la performance [Samoa, 2000] : «

- La sécurité est le contrôle d'accès au composant et la communication chiffrée entre certains composants.
- Les transactions peuvent être vues comme un cas particulier de la propriété de tolérance aux pannes.
- La tolérance aux pannes qui est la réplication d'un composant et la politique de mise en cohérence des copies multiples.
- La persistance qui est la sauvegarde de l'état d'un composant en des points de contrôle spécifiés par l'utilisateur ou décidés par le système.
- La performance qui fait référence à des aspects quantitatifs de qualité de service, tels que la garantie d'un temps d'exécution d'un composant, d'une bande passante dans la communication entre composants, ou de la synchronisation entre flots de communication entre composants ».

Le modèle de descripteur de déploiement est utilisé par l'intergiciel à travers le conteneur pour la gestion des propriétés non fonctionnelles des composants de l'application.

Dans l'approche à composants .NET, le concept de descripteur de déploiement qui permet de définir les propriétés non fonctionnelles sont définies directement dans le code. La définition de ses propriétés se fait dans la phase de développement en utilisant des services COM+ appropriés (*remote access, security, transactional policy, component instances management, synchronization*). L'inconvénient de cette politique figée est qu'un composant une fois implémenté ne peut plus être utilisé dans différents contextes (exemple : le déploiement multiple d'une même application avec différentes politiques de sécurité).

Dans les approches à composants EJB, CCM et Fractal, le méta-modèle du descripteur de déploiement contient les mêmes informations que le méta-modèle d'application, respectivement pour EJB, CCM, Fractal, les méta-modèles d'application (ou méta-modèles du descripteur de déploiement) sont *ejb-jar.DTD*, *CORBAComponentModel.DTD*, *FractalADL.DTD*. Le désavantage majeur qui en découle de la confusion entre le modèle d'application et le modèle de descripteur de déploiement, est qu'une même application ne peut pas être déployée de différentes manières c'est-à-dire avec des propriétés non-fonctionnelles différentes.

Nous proposons donc un méta-modèle du descripteur de déploiement représenté par la figure 16 qui permet de décrire l'architecture logicielle et les propriétés non-fonctionnelles d'une application. Ce méta-modèle ne permet pas de décrire les

contraintes logicielles et les contraintes matérielles. Les contraintes seront décrites au travers du méta-modèle d'application.

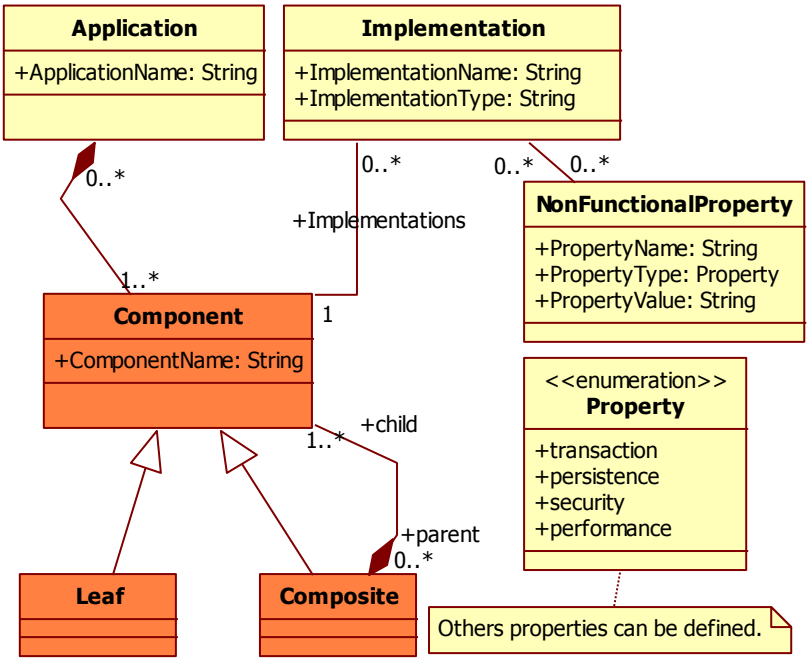


Figure 16: Le méta-modèle du descripteur de déploiement UDeploy

Le modèle de descripteur de déploiement présenté ci-dessous décrit une application scrop composée de deux composants *article* et *localisation*. La persistance du composant *article* est gérée via le *container*.

```
<deploymentdescriptormodel>
   <applicationName>scorp</applicationName>

   <components>
      <component>
         <componentName>article</componentName >
         <implementationName>article-ejb-v3.jar</implementationName >
      </component>
      <component>
         <componentName>localisation</componentName >
         <implementationName>localisation-ccm-v3.jar</implementationName>
      </component>
   </components>

   <nonFunctionalProperties>
      <nonFunctionalProperty>
         <propertyName>P1</propertyName >
         <propertyType>Persistence</propertyType>
         <propertyValue>Container</propertyValue>
      </nonFunctionalProperty>
      …
   </nonFunctionalProperties>

   <properties>
      <property>
         <implementationName>article-ejb-v3.jar</implementationName>
         <propertyNames>
                 <propertyName>P1</propertyName>
         </propertyNames>
      </property>
   </properties>
…
```

```
</deploymentdescriptormodel>
```

5

LA TRANSFORMATION DE MODELES ET LA PLANIFICATION ORIENTEE STRATEGIES

5.1 LA PROPOSITION D'UN LANGAGE DE TRANSFORMATION DE MODELES

Une transformation de modèles [Bézivin et al., 2006] peut être opérée par un langage non formel, par un QVT spécifique ou par un algorithme de transformation qui établit le *mapping* entre les différents modèles. Le langage de transformation que nous proposons est mixte donc basé sur le QVT ATL et sur des algorithmes de transformation (figure 17).

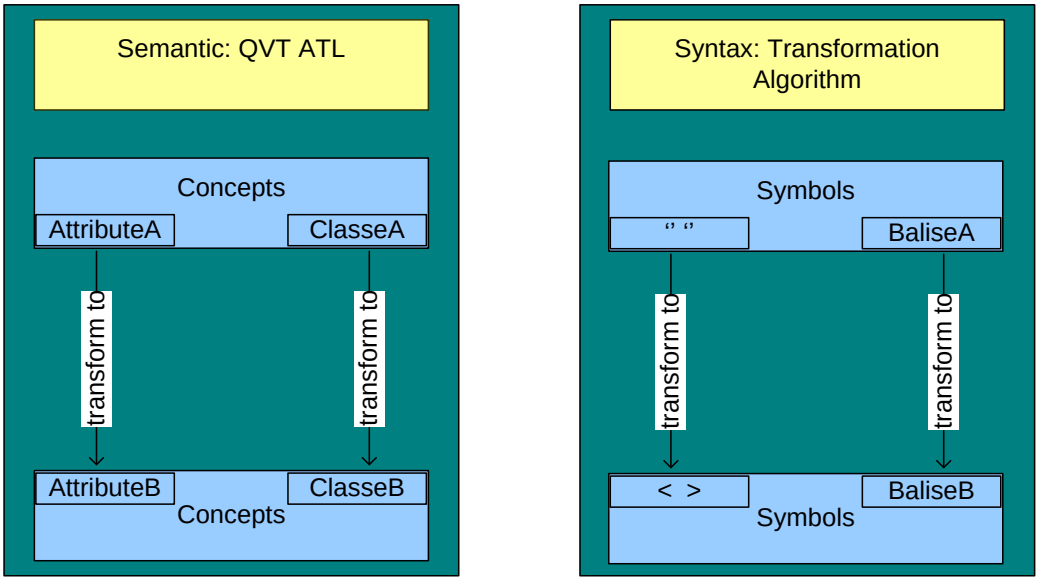


Figure 17 : Le langage de transformation (QVT et algorithme)

La transformation de modèle ne porte ni sur le modèle d'application UDeploy, ni sur le modèle de domaine UDeploy, ni sur le modèle de stratégies UDeploy mais porte sur le modèle de plan de déploiement UDeploy et sur le modèle de descripteur de déploiement UDeploy.

La transformation du modèle de plan de déploiement consiste à la projection du modèle de plan UDeploy au niveau PIM vers des modèles de plan au niveau PSM (EJB, CCM, .NET, SOFA). Les modèles de plan de déploiement spécifiques sont exécutés par les intergiciels cibles afin de mettre en œuvre le déploiement.

La transformation du modèle de descripteur de déploiement consiste à la transformation du modèle de descripteur UDeploy au niveau PSM vers des modèles de descripteur au niveau PIM (EJB, CCM). Les modèles de descripteur de déploiement spécifiques sont utilisées par les intergiciels cibles pour la gestion des composants.

5.1.1 QVT ATL

Nous utilisons le QVT ATL pour la transformation sémantique (Figure 18). La transformation sémantique correspond à la transformation des concepts. Un concept A dans un modèle source peut s'appeler concept B dans un modèle cible. ATL est un langage de transformation de modèles développé au dessus de la plate-forme *Eclipse*. Il fournit des moyens pour produire des modèles cibles à partir des modèles sources via des règles de transformation. Une règle de transformation ATL s'écrit comme suit :

```
rule R {
    from e : nom-méta-entrée ! el-e (cond)
    to s : nom-méta-sortie ! el-s
    ( -- séquence d'attribution de valeurs pour peupler l'élément créé
    -- ex. title<- e.title, nom<- e.nom+"nouveau)
}
```

La règle de nom **R** prend comme métamodèle d'entrée **nom-méta-entrée**. **el-e** est le nom d'un élément du métamodèle. Les instances de cet élément subiront à tour de rôle l'application de la règle **R**. **e** est le nom de la variable locale référençant l'instance de **el-e** (en cours de traitement par la règle **R**) rencontrée dans le modèle d'entrée. **cond** est la condition de filtrage sur les instances d'entrée de la règle. Le nom du métamodèle de sortie est **nom-méta-sortie** et **s** est le nom de la variable locale référençant l'instance créée en sortie de la règle. **el-s** est le nom de l'élément du métamodèle dont il faut créer une instance dans le modèle de sortie, chaque fois qu'une instance d'entrée nommée **e** est rencontrée.

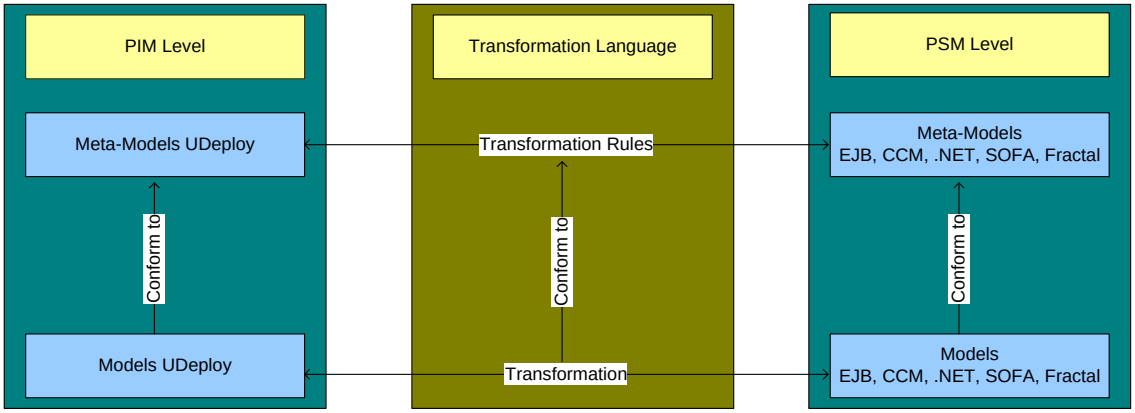


Figure 18 : La transformation de modèle

5.1.2 Algorithmes de transformation

Nous utilisons des algorithmes pour la transformation syntaxique. Un modèle M1 conforme à un méta-modèle source peut être en écrit un Java tandis qu'un modèle M2 conforme à un méta-modèle cible peut être écrit en XML.

Les méta-modèles du plan de déploiement UDeploy et du descripteur de déploiement UDeploy sont écrits en DTD (Document Type Definition). Pour des raisons pratiques, nous avons décidé de développer nos algorithmes et de gérer la persistance des modèles avec Java. Ainsi nous avons opérer trois transformations de base (figure 19) :

- La transformation des méta-modèles UDeploy DTD vers des méta-modèles UDeploy XSD via l'outil XMLPad.
- La transformation des méta-modèles UDeploy XSD vers des méta-modèles UDeploy Ecore via l'outil EMF.
- La transformation des méta-modèles Ecore vers des méta-modèles UDeploy Java via l'outil EMF.

La chaine de transformations du méta-modèle de plan DTD et du méta-modèle du descripteur DTD vers le méta-modèle de plan Java et le méta-modèle du descripteur Java ne se fait qu'une seule fois.

Une fois les classes Java créées, elles seront instanciées par des données du plan de déploiement et du descripteur de déploiement.

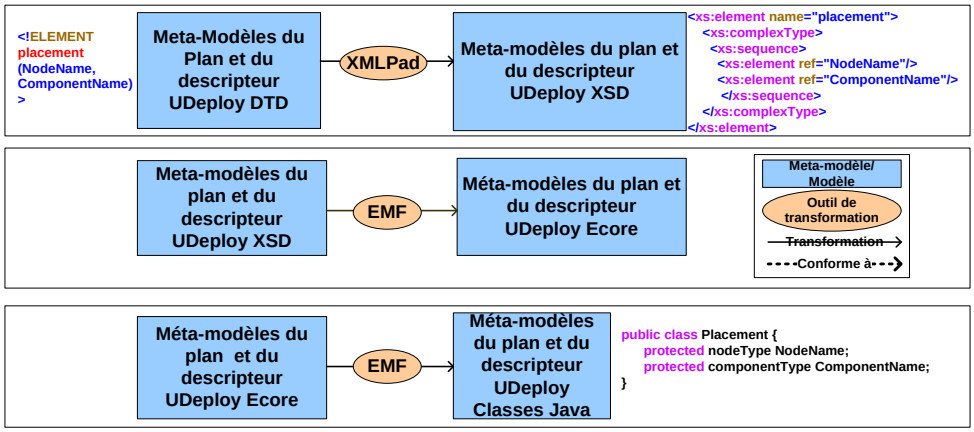


Figure 19 : La transformation syntaxique

Nous avons des algorithmes de transformation syntaxique pour chaque technologie comme EJB (*AlgoEJBPlan*, *AlgoEJBDescripteur*), CCM (AlgoCCMPlan, *AlgoCCMDescripteur*), .NET(*AlgoNETPlan*) et SOFA (*AlgoSOFAPlan*). L'algorithme permet de produire un modèle de sortie qui sera conforme au méta-modèle de sortie syntaxiquement. Dans la section 5.2, nous décrivons les algorithmes de transformation du plan de déploiement pour différentes technologies, les algorithmes de

transformations du descripteur de déploiement sont basés sur les mêmes principes.

5.2 DES EXEMPLES DE PERSONNALISATION SEMANTIQUE VIA LE QVT ATL

5.2.1 La personnalisation du plan de déploiement EJB, .NET et CCM

A la fin du processus de planification nous obtenons un modèle de plan de déploiement UDeploy qui est au niveau PIM. Ce plan de déploiement doit être personnalisé pour des plateformes cibles d'exécution.

L'exemple ci-dessous montre le processus de transformation du méta-modèle du plan de déploiement UDeploy vers le méta-modèle du plan des plateformes EJB, .NET et CCM.

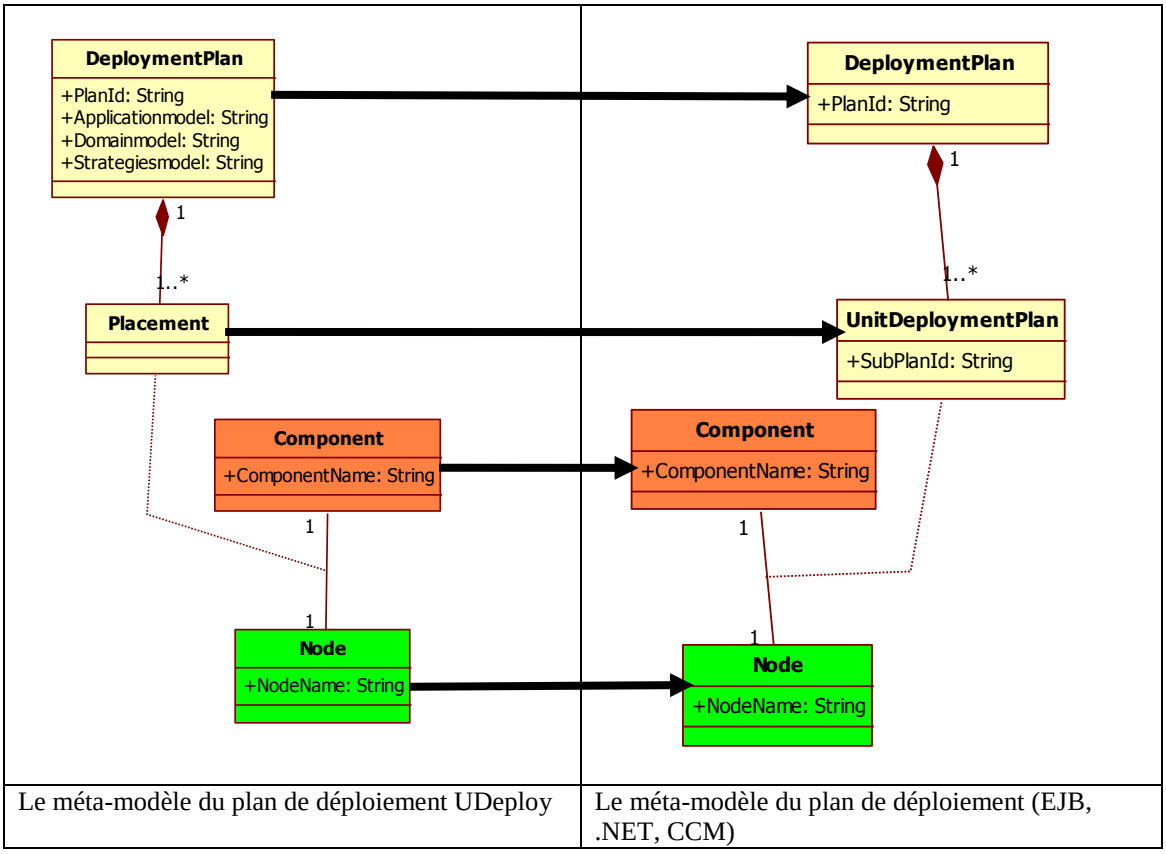


Figure 20 : La personnalisation du plan de déploiement (MM UDeploy vers MM EJB, .NET, CCM)

La règle 1 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement EJB, .NET et CCM (cible). La transformation porte sur la classe *DeploymentPlan* du méta-modèle source et sur la classe *DeploymentPlan* du méta-modèle cible. L'attribut *PlanId* du méta-modèle cible sera l'attribut *PlanId* du méta-modèle source.

```
rule R1 {
from in : UDeployDeploymentPlanMetaModel ! DeploymentPlan
to out : EJB_NET_CCMDeploymentPlanMetaModel ! DeploymentPlan
        PlanId<- in.PlanId }
```

La règle 2 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement EJB, .NET et CCM (cible). La transformation porte sur la classe *DeploymentPlan* du méta-modèle source et sur la classe *UnitDeploymentPlan* du méta-modèle cible. L'attribut *SubPlanId* de la classe *UnitDeploymentPlan* sera une concaténation de l'attribut *PlanId* du modèle source et d'un numéro de plan fourni par l'utilisateur *getSubPlanNmber().*

```
rule R2 {
from in : UDeployDeploymentPlanMetaModel ! DeploymentPlan
to out : EJB_NET_CCMDeploymentPlanMetaModel ! UnitDeploymentPlan
SubPlanId<- in.PlanId + getSubPlanNmber()}
```

La règle 3 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement EJB, .NET et CCM (cible). La transformation porte sur la classe *component* du méta-modèle source et sur la classe *component* du méta-modèle cible. L'attribut *ComponentName* du méta-modèle cible sera l'attribut *ComponentName* du méta-modèle source.

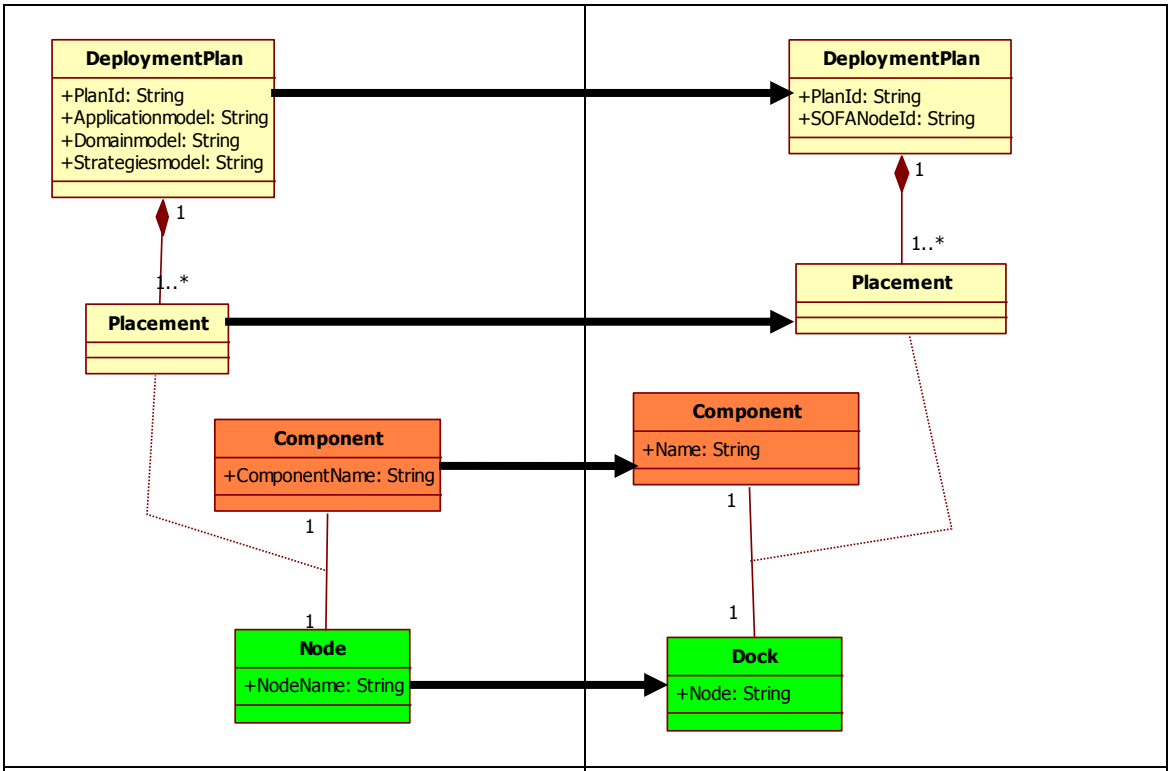
```
rule R3 {
from in : UDeployDeploymentPlanMetaModel ! Component
to out : EJB_NET_CCMDeploymentPlanMetaModel ! Component
ComponentName<- in.ComponentName }
```

La règle 4 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement EJB, .NET et CCM (cible). La transformation porte sur la classe *Node* du méta-modèle source et sur la classe *Node* du méta-modèle cible. L'attribut *NodeName* du méta-modèle cible sera l'attribut *NodeName* du méta-modèle source.

```
rule R4 {
from in : UDeployDeploymentPlanMetaModel ! Node
to out : EJB_NET_CCMDeploymentPlanMetaModel ! Node
NodeName<- in.NodeName }
```

5.2.2 La personnalisation du plan de déploiement SOFA

L'exemple ci-dessous montre le processus de transformation du méta-modèle du plan de déploiement UDeploy vers le méta-modèle du plan SOFA.



Le méta-modèle du plan de déploiement UDeploy	Le méta-modèle du plan de déploiement SOFA

La règle 1 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement SOFA (cible). La transformation porte sur la classe *DeploymentPlan* du méta-modèle source et sur la classe *DeploymentPlan* du méta-modèle cible. L'attribut *PlanId* du méta-modèle cible sera l'attribut *PlanId* du méta-modèle source et l'attribut *SOFANodeId* sera fourni par l'utilisateur.

```
rule R1 {
from in : UDeployDeploymentPlanMetaModel ! DeploymentPlan
to out : SOFADeploymentPlanMetaModel ! DeploymentPlan
        PlanId<- in.PlanId,
        SOFANodeId<- getSofaNodeId()}
```

La règle 2 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement SOFA (cible). La transformation porte sur la classe *component* du méta-modèle source et sur la classe *component* du méta-modèle cible. L'attribut *Name* du méta-modèle cible sera l'attribut *ComponentName* du méta-modèle source.

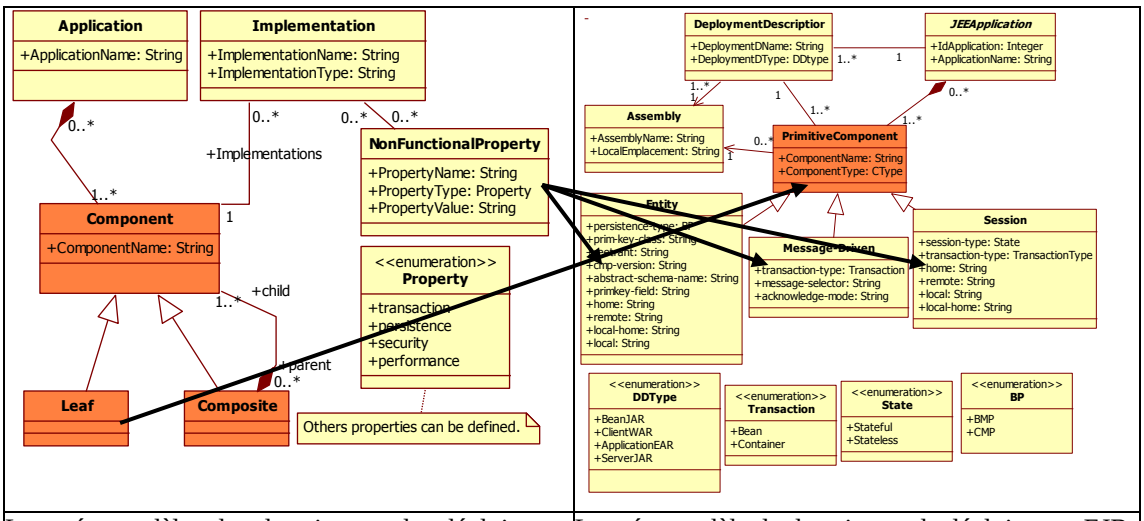
```
rule R2 {
from in : UDeployDeploymentPlanMetaModel ! Component
to out : SOFADeploymentPlanMetaModel ! Component
Name<- in.ComponentName }
```

La règle 3 prend en entrée le méta-modèle du plan de déploiement UDeploy (source) et produit en sortie un méta-modèle du plan de déploiement SOFA (cible). La transformation porte sur la classe *Node* du méta-modèle source et sur la classe *Dock* du méta-modèle cible. L'attribut *Node* du méta-modèle cible sera l'attribut *NodeName* du méta-modèle source.

```
rule R4 {
from in : UDeployDeploymentPlanMetaModel ! Node
to out : SOFADeploymentPlanMetaModel ! Dock
Node<- in.NodeName }
```

5.2.3 La personnalisation du descripteur de déploiement EJB

L'exemple ci-dessous montre le processus de transformation du méta-modèle du descripteur de déploiement UDeploy vers celui de la plate-forme EJB.



Le méta-modèle du descripteur de déploiement UDeploy	Le méta-modèle du descripteur de déploiement EJB

Figure 21 : La personnalisation du descripteur de déploiement (MM UDeploy vers MM EJB)

La règle 1 prend en entrée le méta-modèle du descripteur de déploiement UDeploy (source) et produit en sortie un méta-modèle du descripteur de déploiement EJB (cible). La transformation porte sur la classe *leaf* du méta-modèle (implémentation *EJB ENTITY*) et sur la classe *PrimitiveComponent* du méta-modèle cible.

```
rule R1 {
from in : UDeployDeploymentDescriptorMetaModel ! Leaf
        (Component.Implementation.ImplementationType = EJB ENTITY)
to out : EJBDeploymentDescriptorMetaModel ! PrimitiveComponent
        ComponentName<- in.Component.ComponentName,
        ComponentType <- "EJB ENTITY",
        persistence-type <-getPersistence(),
        prim-key-class <-getPrimaryKey(),
        reetrant<-getReetrant,
        cmp-version<-getcmpversion(),
        abstract-schema-name<-getabstractschemaname(),
        home<-gethome(),
        remote<-getremote(),
        local-home<-getlocalhome(),
        local<-getlocal()}
```

La règle 2 prend en entrée le méta-modèle du descripteur de déploiement UDeploy (source) et produit en sortie un méta-modèle du descripteur de déploiement EJB (cible). La transformation porte sur la classe *leaf* du méta-modèle (implémentation *EJB SESSION*) et sur la classe *PrimitiveComponent* du méta-modèle cible.

```
rule R2 {
from in : UDeployDeploymentDescriptorMetaModel ! Leaf
        (Component.Implementation.ImplementationType = EJB SESSION)
to out : EJBDeploymentDescriptorMetaModel ! PrimitiveComponent
        ComponentName<- in.Component.ComponentName,
        ComponentType <- "EJB SESSION",
        transaction-type <-getTransaction(),
        session-type <-getSessionType(),
        home<-gethome(),
        remote<-getremote(),
        local<-getlocal(),
        local-home<-getlocalhome()}
```

La règle 3 prend en entrée le méta-modèle du descripteur de déploiement UDeploy (source) et produit en sortie un méta-modèle du descripteur de déploiement EJB (cible). La transformation porte sur la classe *leaf* du méta-modèle (implémentation *EJB MESSAGE DRIVEN*) et sur la classe *PrimitiveComponent* du méta-modèle cible.

```
rule R3 {
from in : UDeployDeploymentDescriptorMetaModel ! Leaf
        (Component.Implementation.ImplementationType = EJB MESSAGE-DRIVEN)
to out : EJBDeploymentDescriptorMetaModel ! PrimitiveComponent
        ComponentName<- in.Component.ComponentName,
        ComponentType <- "EJB MESSAGE-DRIVEN",
        transaction-type <-getTransaction(),
        message-selector <-getmessageselector(),
        aknowledge-mode<-getaknowledgemode()}
```

5.3 DES EXEMPLES DE PERSONNALISATION SYNTAXIQUE VIA LES ALGORITHMES DE TRANSFORMATION

Nous présentons dans ce qui suit quatre exemples de personnalisation syntaxique (Figure 22). Les algorithmes de personnalisation du plan de déploiement pour les plateformes EJB, CCM, .NET et SOFA sont respectivement AlgoEJBPlan, AlgoCCMPlan, AlgoNETPlan et AlgoSOFAPlan.

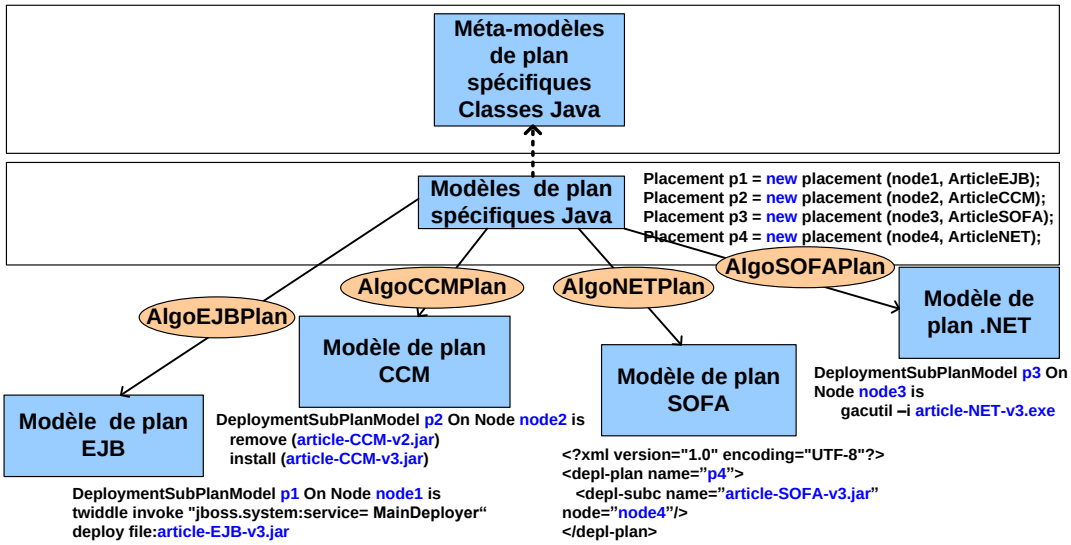


Figure 22 : Les algorithmes de transformation

5.3.1 La personnalisation du plan de déploiement EJB

```
Entrée : Modèle de plan spécifique EJB mEJB
Sortie : document d
Debprog ;
        Créer un document d ;
        Pour chaque placement p du modèle de plan mEJB Faire
                C= RecupererNomComposant(p) ;
                IC= RécupererNomImplementation(C);
                N= RecupererNomNode(p) ;
                NT= RecupererNomServeur(N) ;
                Si (NT== JBOOS) alors d.EcrireDansDocument ('On Node', N , 'is twiddle
                invoke "jboss.system:service= MainDeployer" deploy file:',IC);
                Finsi;

                Sinon SI (NT==JONAS) alors d.EcrireDansDocument('On Node', N , 'jonas
                admin -a',IC) ;
                Finsinon ;

        Finfaire ;
```

```
        Retourner le document d ;
Finprog ;
```

5.3.2 La personnalisation du plan de déploiement .NET

```
Entrée : Modèle de plan spécifique .NET mNET
Sortie : document d
Debprog ;
        Créer un document d
        Pour chaque placement p du modèle de plan Mnet Faire
                C= RecupererNomComposant(p) ;
                IC= RécupererNomImplementation(C) ;
                N= RecupererNomNode(p) ;
                d.EcrireDansDocument ('On Node', N, ' is gacutil -i',IC);
        Finfaire ;
        Retourner le document d ;
Finprog ;
```

5.3.3 La personnalisation du plan de déploiement CCM

```
Entrée : Modèle de plan spécifique CCM mCCM
Sortie : document d
Debprog ;
        Créer un document d
        Pour chaque placement p du modèle de plan mCCM Faire
                C= RecupererNomComposant(p) ; IC= RécupererNomImplementation(C) ;
                N= RecupererNomNode(p) ;
                d.EcrireDansDocument ('On Node', N, 'Install(',IC, ')');
        Finfaire ;
        Retourner le document d ;
Finprog ;
```

5.3.4 La personnalisation du plan de déploiement SOFA

```
Entrée : Modèle de plan spécifique SOFA mSOFA
Sortie : document xml d
Debprog ;
        Créer un document xml d
        NP= RecupererNomComposant(MSOFA) ;
        d.EcrireDansDocument (' <?xml version="1.0" encoding="UTF-8"?> ');
        d.EcrireDansDocument ('<depl-plan name="p4">', NP, ' "> ') ;
        Pour chaque placement p du modèle de plan mSOFA Faire
                C= RecupererNomComposant(p) ; IC= RécupererNomImplementation(C) ;
                N= RecupererNomNode(p) ;
                d.EcrireDansDocument (' <depl-subc name=" ', IC, ' " node=" ',N, '" > ');
        Finfaire ;
        Retourner le document d ;
```

```
Finprog ;
```

5.4 LA PLANIFICATION

Le calcul du plan de déploiement se fait au travers d'un planificateur. Le planificateur (figure 23) prend en entrée un modèle d'application, un modèle de domaine et un modèle de stratégies de déploiement.

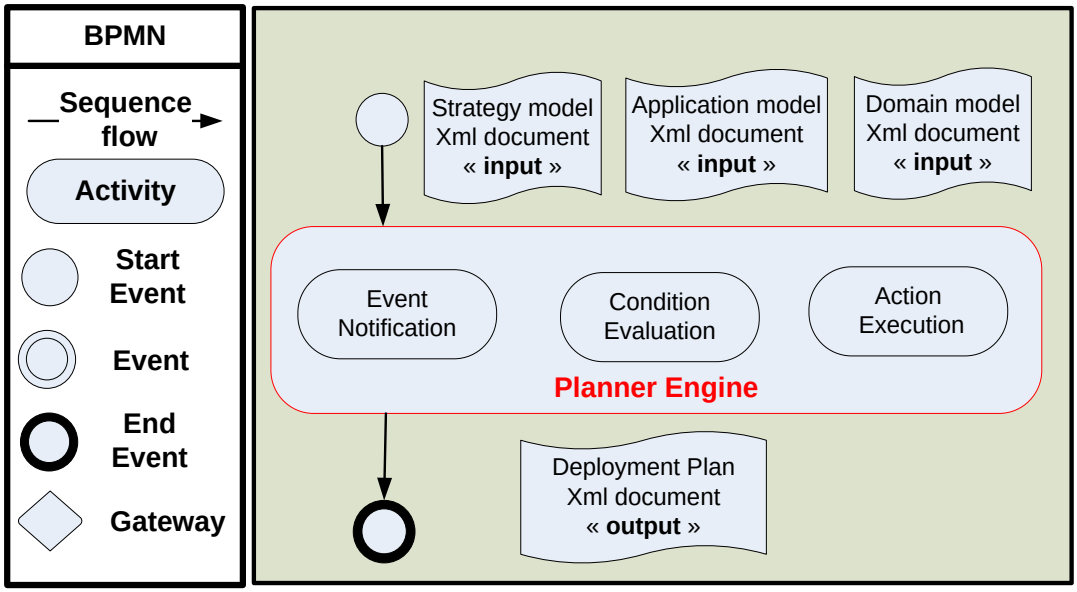


Figure 23 : La création du plan de déploiement

Le planificateur effectue un calcul de compatibilité entre les ressources disponibles du domaine, les besoins en ressources de l'application à déployer et les stratégies définies. Ainsi, le plan de déploiement pour une application A composée de C1 à Ci composants où i>=1 et pour un domaine D formé de N1 à Nj nœuds où j>=1, est l'ensemble des placements valides (Ci, Nj). Voir en annexe 1 l'algorithme de planification.

6

LA DESCRIPTION DU PROTOTYPE

6.1 LE PROTOTYPE UDEPLOY

Le système UDeploy [Dibo and Belkhatir, 2010b] est structuré en trois niveaux : la couche interface, l'applicatif et les données.

Les interfaces graphiques. Les interfaces sont le pont entre l'utilisateur et le système. Le déploiement est un processus complexe puisqu'il demande de nombreuses interactions entre l'utilisateur et la machine. Ainsi pour limiter les erreurs et rendre l'interaction plus facile, nous avons défini des interfaces ergonomiques basées sur les critères de Scapin et Bastien [Bastien and Scapin, 1993] [Bastien et al., 1996]. Leur travail d'ergonome s'articule autour d'outils qui leur permettent de juger de l'utilité et de l'utilisabilité d'un système informatique. Parmi ces outils, il existe de nombreuses normes, recommandations et check-lists visant à fournir un cadre à l'expertise de l'ergonome. Face à la multitude des recommandations existantes, Bastien et Scapin ont proposé à partir de la synthèse d'environ 900 recommandations dans le domaine de l'ergonomie informatique au sens large, une liste de 18 critères répartis en 8 dimensions [Bastien et al., 1999]. Les critères pris en compte dans l'environnement UDeploy sont le guidage, la charge de travail, l'adaptabilité et la gestion des erreurs.

L'applicatif. Nous avons quatre modules principaux qui sont la modélisation, la transformation, le calcul et l'exécution :

- La modélisation de l'application, du domaine et des stratégies de déploiement se fait au travers d'EMF et la persistance des données se fait sous PostgreSQL. Il y a une particularité pour la modélisation du domaine. La mise-à-jour du modèle de domaine est fait via des outils comme Sun Grid Engine [Engine, 2009], Globus [Globus, 2009] et Condor [Condor, 2009] qui permettent d'accéder aux ressources d'un domaine et de connaitre les ressources disponibles des nœuds, de connaitre les nœuds disponibles et ceux indisponibles.
- La transformation de modèle est faite au travers du QVT ATL et des algorithmes de transformation.
- Le calcul du plan est basé sur un algorithme de planification.
- L'exécution du plan utilise un protocole FTP pour cela nous avons utilisé la bibliothèque ftpBean.

Les données. La persistance des données est gérée sous PostgreSQL.

Tableau 3 : Architecture technique de UDeploy

Niveaux	Langages, Environnements, Bibliothèques
Interface	Java, Bibliothèque Swing
Applicatif	Éditeur xml, Java, PostgresJDBC, EMF, QVT ATL, Sun Grid Engine, ftpBean
Données	PostgreSQL

6.2 L'ARCHITECTURE FONCTIONNELLE

UDeploy permet de réaliser les fonctionnalités décrites sur le diagramme de cas d'utilisation représenté par la figure 24. Il permet :

- de se connecter aux différentes bases « *repository* »,
- de consulter, de créer et de modifier un modèle d'application, un modèle de domaine ou un modèle de stratégies de déploiement,
- de calculer un modèle de plan de déploiement,
- de personnaliser le modèle de plan de déploiement et le modèle de descripteur de déploiement,
- d'exécuter le modèle de plan de déploiement personnalisé.

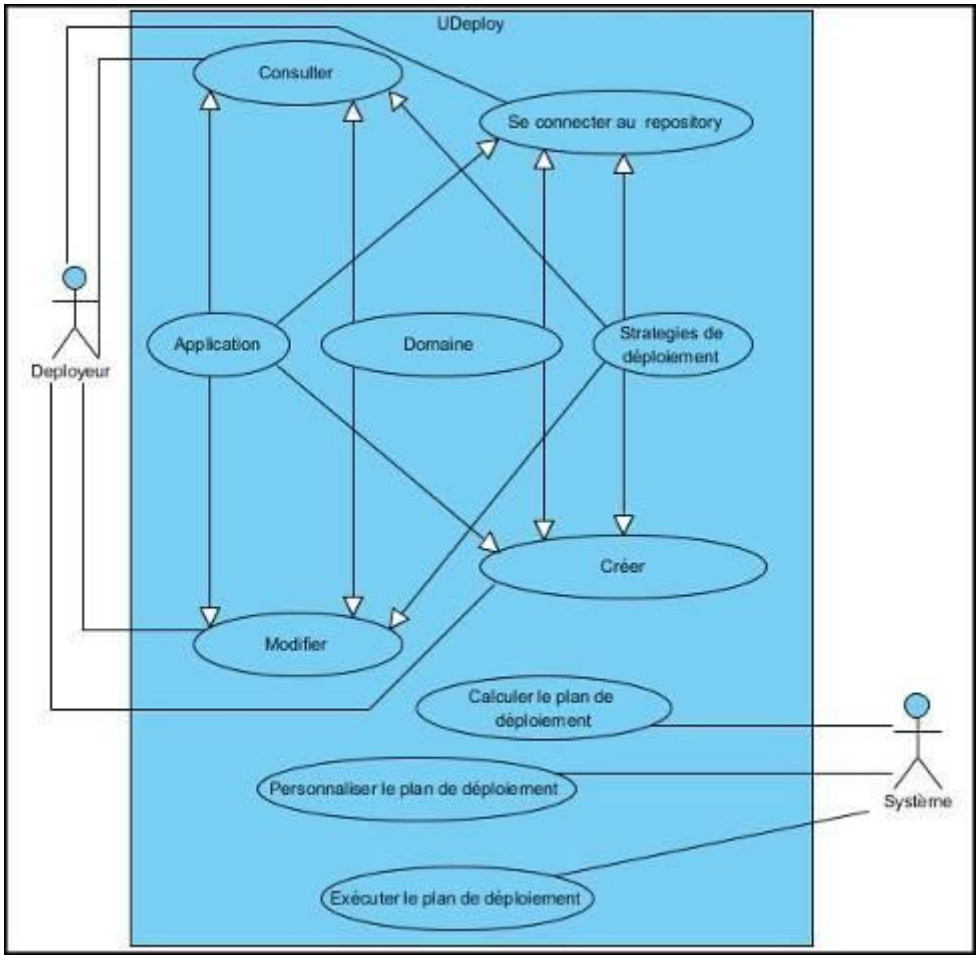


Figure 24 : Le diagramme de cas d'utilisation de UDeploy

Nous allons décrire dans ce qui suit les différentes activités de déploiement supportées par UDeploy. Nous décrirons en détail 1) la connexion à la base d'application, la consultation, la création et la modification d'un modèle d'application, 2) le calcul d'un modèle de plan de déploiement, 3) la personnalisation du modèle de

plan de déploiement et du modèle de descripteur de déploiement et 4) l'exécution du modèle de plan de déploiement.

Les activités de connexion, de consultation, de création et de modification pour le domaine et les stratégies de déploiement seront décrites de façon non détaillée. En effet l'application, le domaine et les stratégies peuvent être vus comme des concepts similaires : l'application est composée de composants, le domaine est composé de sites et les stratégies de déploiement sont composées de règles. De ce fait nous avons défini une fonction générique pour chaque activité (la fonction connexion, consultation, création et modification) et nous les avons adaptés à chaque concept.

La figure 25 montre l'interface graphique principale de l'outil UDeploy.

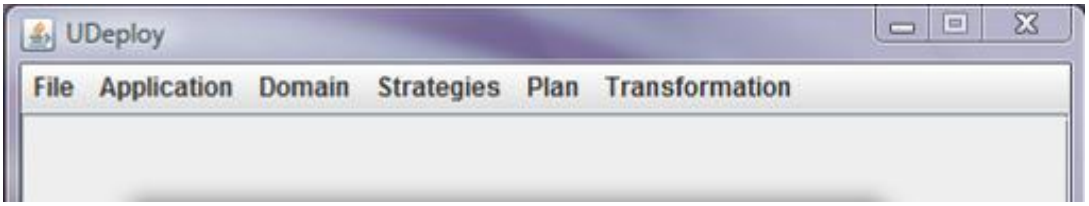


Figure 25 : L'interface principale de UDeploy

6.2.1 La modélisation de l'application, du domaine et des stratégies de déploiement

Les figures 26, 27 et 28 montrent respectivement les tâches pouvant être accomplies pour la modélisation d'une application (connexion, consultation, création et mise-à-jour des composants), pour la modélisation d'un domaine (connexion, consultation, création et mise-à-jour des nœuds), et pour la modélisation des stratégies de déploiement (connexion, consultation, création et mise-à-jour des règles ECA).

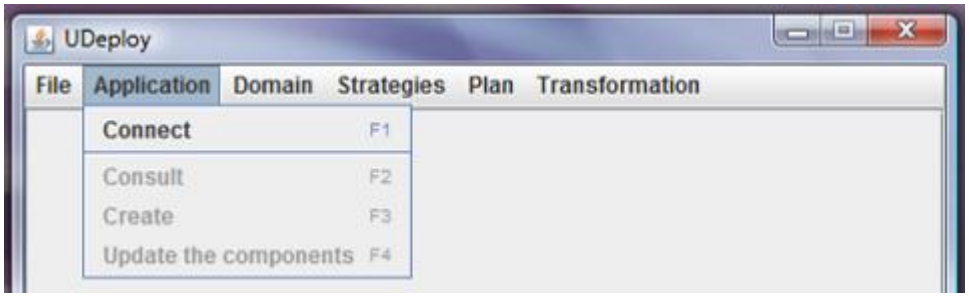


Figure 26 : Les fonctions disponibles pour la manipulation de l'application

Figure 27 : Les fonctions disponibles pour la manipulation du domaine

Figure 28 : Les fonctions disponibles pour la manipulation des stratégies

a. La connexion à la base d'application

Pour se connecter à la base d'application, le système demande l'adresse de localisation de la base de données, un identifiant et un mot de passe. Une fois ces informations transmises, le système crée une connexion si l'identification est valide, sinon il affiche un message d'erreur.

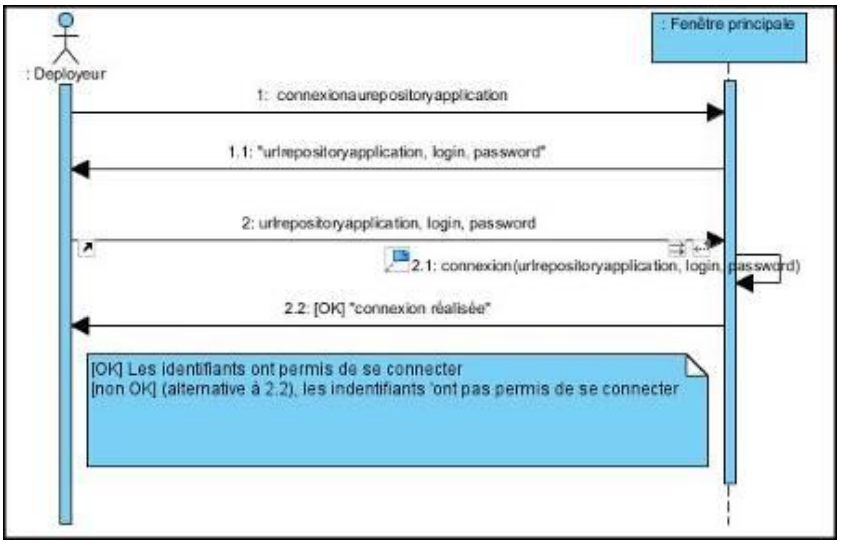


Figure 29 : Le diagramme de séquence pour la connexion

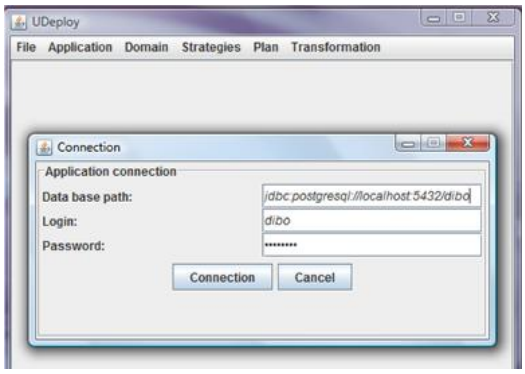


Figure 30 : L'interface graphique pour la connexion à la base d'application

La connexion se fait de la même manière pour la base du domaine et la base des stratégies.

b. La consultation

Pour consulter un modèle d'application, on propose à l'utilisateur de choisir un modèle d'application parmi la liste des applications disponibles dans la base. L'activité de consultation peut être effectuée si et seulement si la connexion à la base est établie. Par la suite, la fonction affiche tous les composants de l'application ainsi que pour chaque composant tous les besoins logiciels et matériels qu'il requiert.

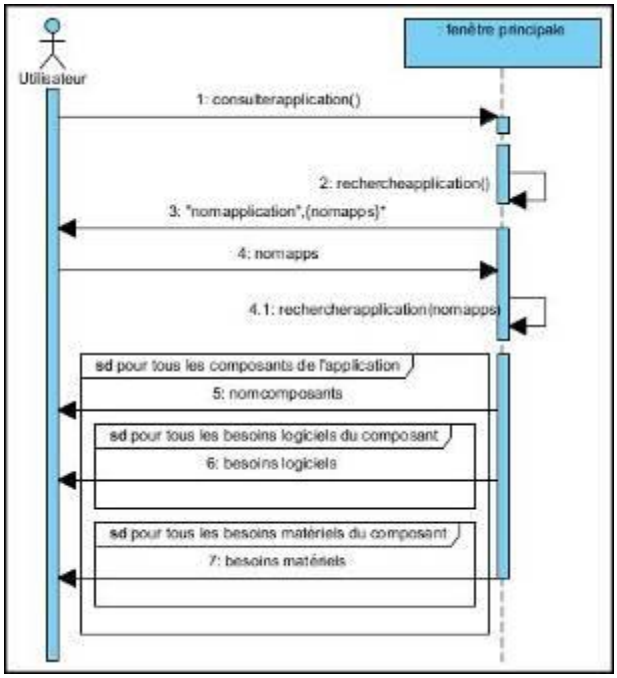


Figure 31 : Le diagramme de séquence pour la consultation d'une application

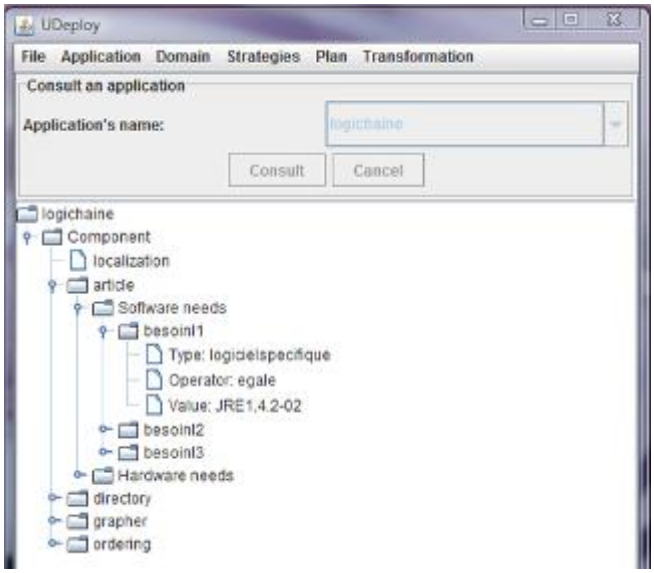


Figure 32 : L'interface pour la consultation d'une application

La consultation du modèle de domaine ou du modèle de stratégies de déploiement se fait de la même manière.

Pour le modèle de domaine, les nœuds ainsi que les ressources logicielles et matérielles qu'ils offrent sont affichés.

Pour le modèle de stratégies de déploiement, les règles ainsi que les événements, les conditions et les actions qui leurs sont associés sont affichés.

c. La création

Pour créer un modèle d'application, l'outil demande via l'interface de la Figure 33 le nom de la future application à l'utilisateur. Puis, les composants présents dans la base d'application sont affichés, l'utilisateur peut les ajouter ou non. L'exécution de cette fonction permet aussi à l'utilisateur de créer de nouveaux composants. Ainsi le composant créé sera ajouté à la liste des composants de la base.

La création d'un composant est assez similaire à la création d'une application car une fois que l'utilisateur a proposé un nom de composant, UDeploy lui propose les contraintes logicielles et matérielles déjà définies dans la base (Figure 34, *Make a list of software requirements*, *Make a list of hardware requirements*). Cette fonction peut aussi entraîner la création de contraintes logicielles et/ou matérielles. Ainsi les contraintes logicielles et/ou matérielles créées seront ajoutées à la liste des contraintes disponibles dans la base.

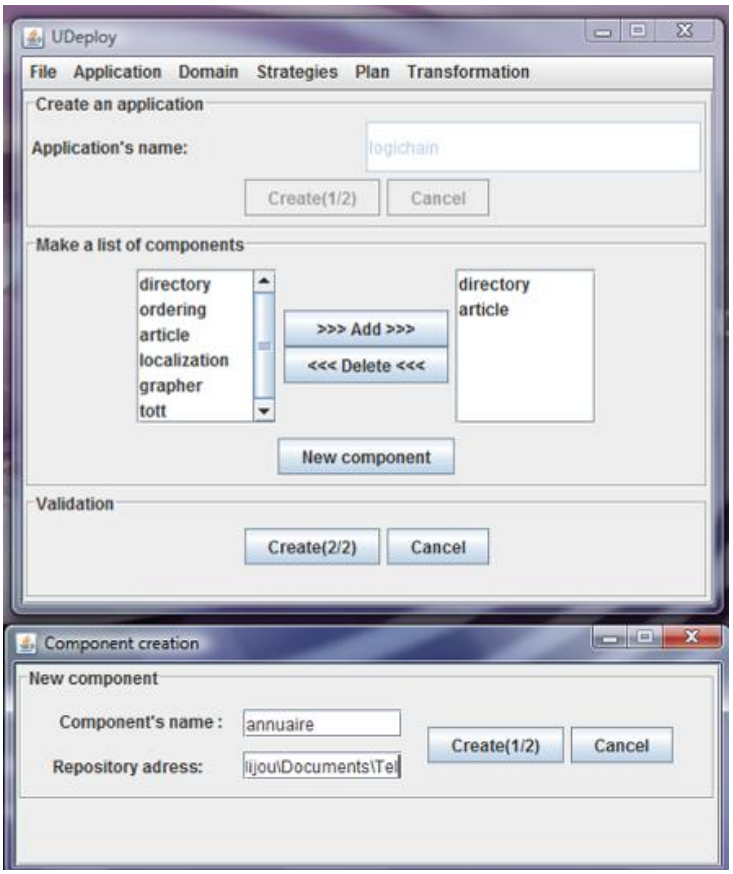


Figure 33: La création d'un modèle d'application

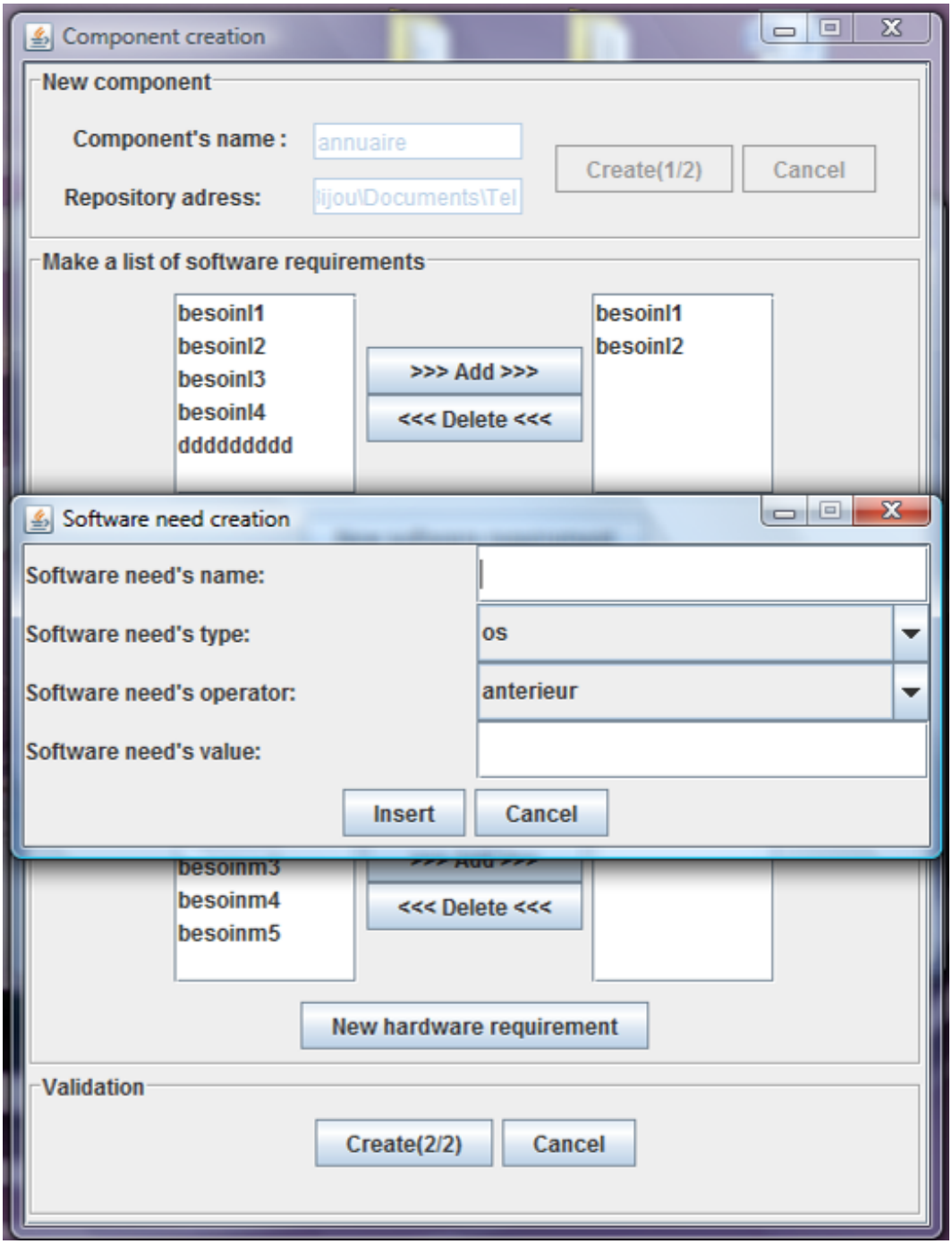


Figure 34 : La création d'un composant et ses contraintes logicielles et matérielles

d. La modification

La modification d'une application combine les opérations de la consultation et de la création d'un modèle d'application. La première partie est identique au début d'une consultation (où l'on propose à l'utilisateur d'identifier l'application à modifier). La seconde partie est très proche de la fonction création d'une application car on doit proposer les composants de la base pour un éventuel ajout (Figure 33 partie *Make a list of components*).

Lors de l'exécution de cette fonction, l'utilisateur pourra créer un nouveau composant (figure 35) ; cela implique que le nouveau composant devra être ajouté à la liste des composants de la base.

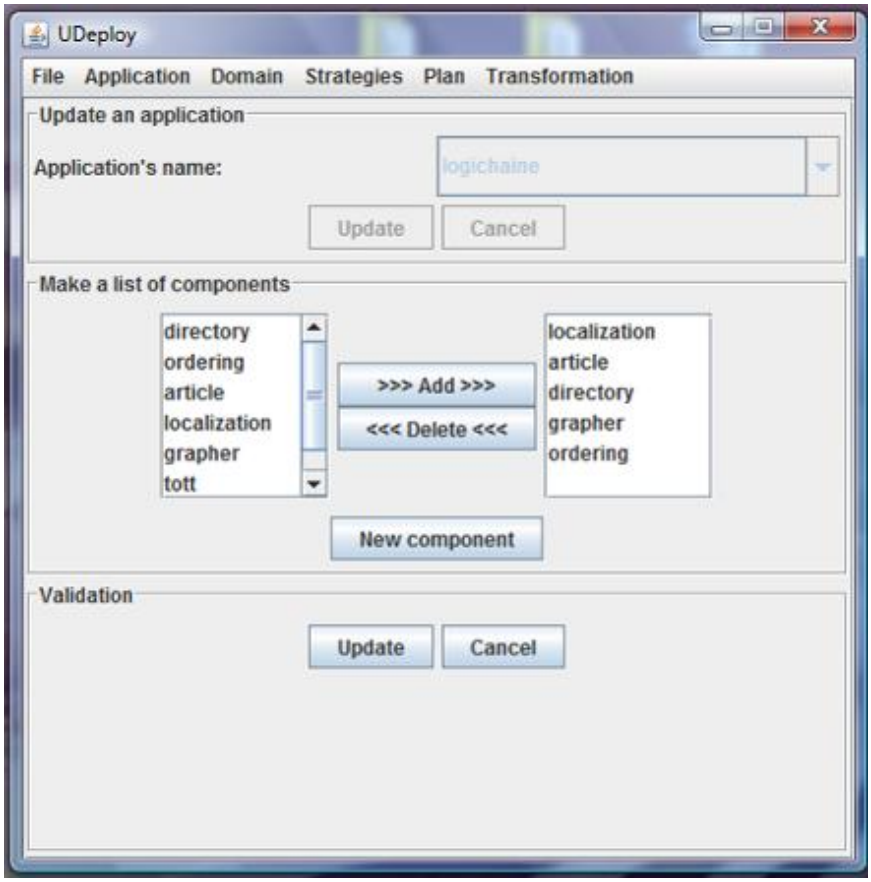


Figure 35 : La modification d'une application

La fonction de modification est adaptée aux concepts de domaine et de stratégies de déploiement.

6.2.2 La création du modèle de plan de déploiement

Les fonctionnalités disponibles pour l'activité de planification sont la connexion à la base du plan, le calcul du plan de déploiement et l'exécution du plan de déploiement. Par contre, l'exécution du plan de déploiement est possible si et seulement si l'activité de transformation est terminée.

Figure 36 : L'interface pour la manipulation du plan

Le plan de déploiement se construit progressivement (Figure 37). Dans notre outil, les fonctions identifiées et l'ordonnancement choisi s'opère comme suit :

1. Demander à l'utilisateur pour quelle modèle d'application, pour quel modèle de domaine et avec quel modèle de stratégies de déploiement, le modèle de plan doit être calculé (Figure 37, processus 1/4).
2. Demander à l'utilisateur quelles sont les contraintes matérielles et logicielles des composants de l'application qui doivent être respectées (Figure 37, processus 2/4).
3. Calculer les affectations possibles entre les composants et les nœuds du domaine suivant l'algorithme de planification (cet algorithme est décrit dans la section 5.4).
4. Demander à l'utilisateur de valider des placements proposés. Lorsque l'utilisateur valide un placement, nous devons ré-effectuer l'étape 3 car les ressources logicielles et matérielles du nœud choisi auront diminuées (Figure 37, processus 3/4).
5. Demander à l'utilisateur de donner un nom au modèle de plan de déploiement calculé (Figure 37, processus 4/4).

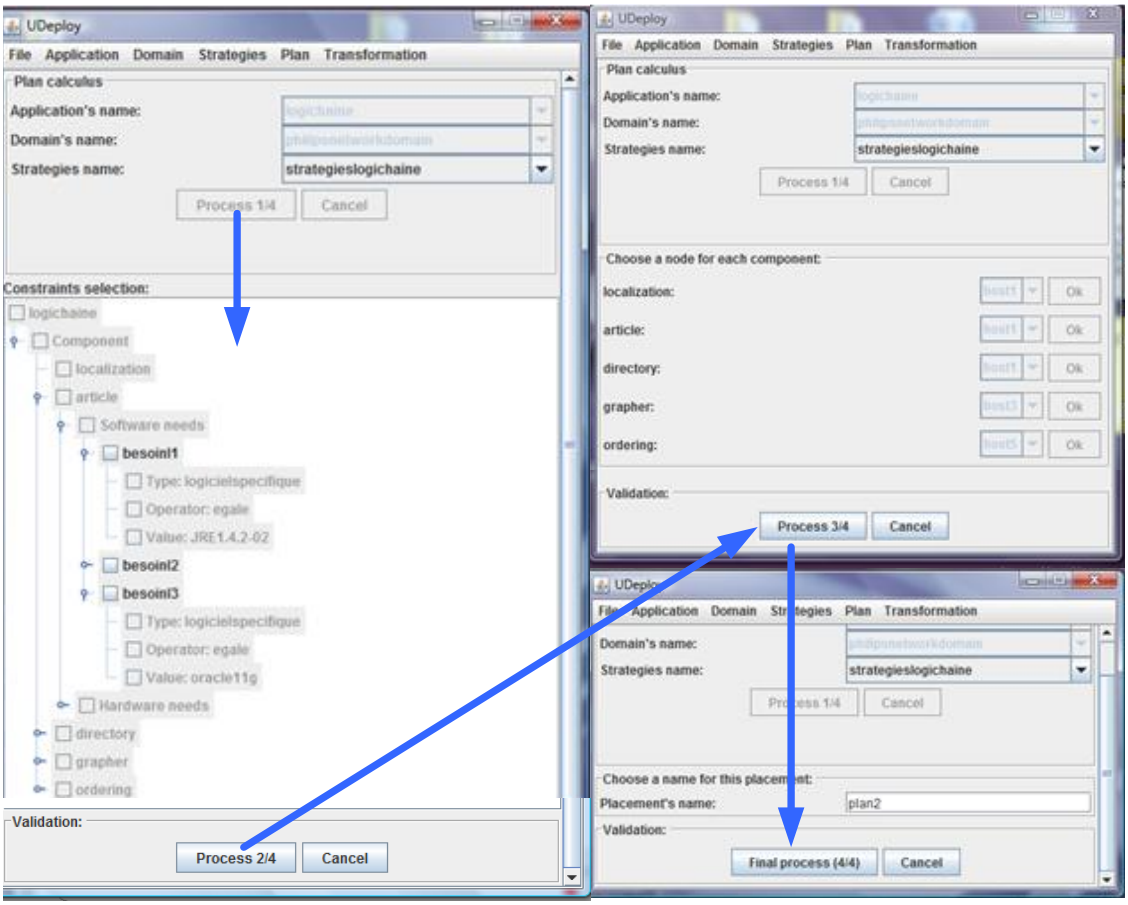


Figure 37 : Processus de création du plan de déploiement

6.2.3 La transformation

L'activité de transformation couvre la génération du modèle de descripteur de déploiement et la personnalisation du modèle de plan de déploiement (Figure 38).

Figure 38 : La transformation

a. La génération du modèle de descripteur de déploiement

Une fois que les classes Java (modèle d'application et données fournies par le *deployeur* sur les propriétés non fonctionnelles de l'application) sont instanciées, nous utilisons ces données pour générer le modèle de descripteur de déploiement. Cette génération correspond à une transformation de modèle.

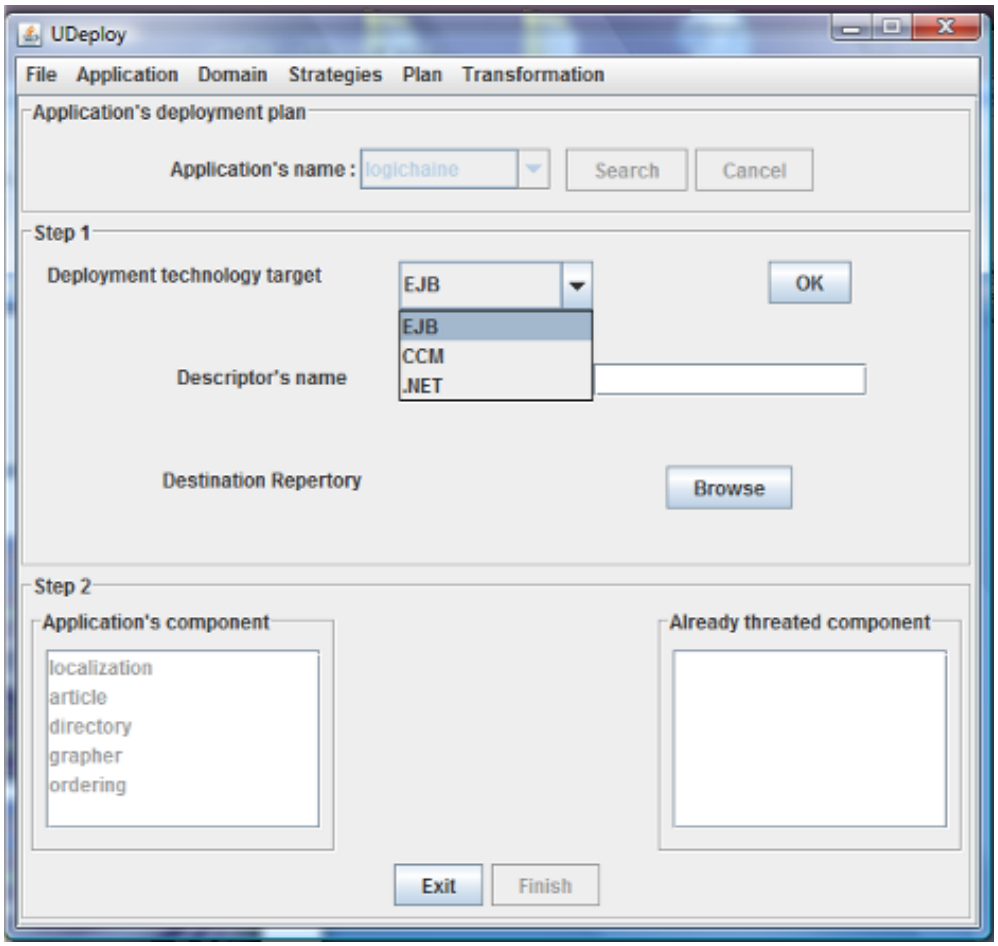


Figure 39 : La fenêtre de saisie des informations générales

Pour générer un modèle de descripteur de déploiement, il faut passer par deux étapes :

– La première est d'entrer les informations générales concernant le modèle de descripteur de déploiement de l'application choisie (Figure 39).

– La deuxième consiste à entrer certaines informations pour chaque composant de l'application qui ne sont pas connues via le modèle d'application (Figure 39).

– Concernant la saisie des informations générales d'un descripteur de déploiement :

  - L'utilisateur choisira une application parmi la liste proposée pour laquelle, il souhaite générer un modèle de descripteur de déploiement.
  - L'utilisateur choisira la technologie cible du descripteur de déploiement.
  - Pour certaines technologies, le descripteur a un nom prédéfini, pour EJB c'est « ejb-jar.xml » mais pour d'autres technologies l'utilisateur choisira un nom.
  - L'utilisateur sélectionnera un dossier de destination du descripteur à l'aide du bouton « browse » (Figure 39).
  - Si l'utilisateur ne choisit pas de dossier de destination alors par défaut le descripteur sera enregistré dans le dossier de chaque composant pour être certain qu'il sera accompagné de ce descripteur.

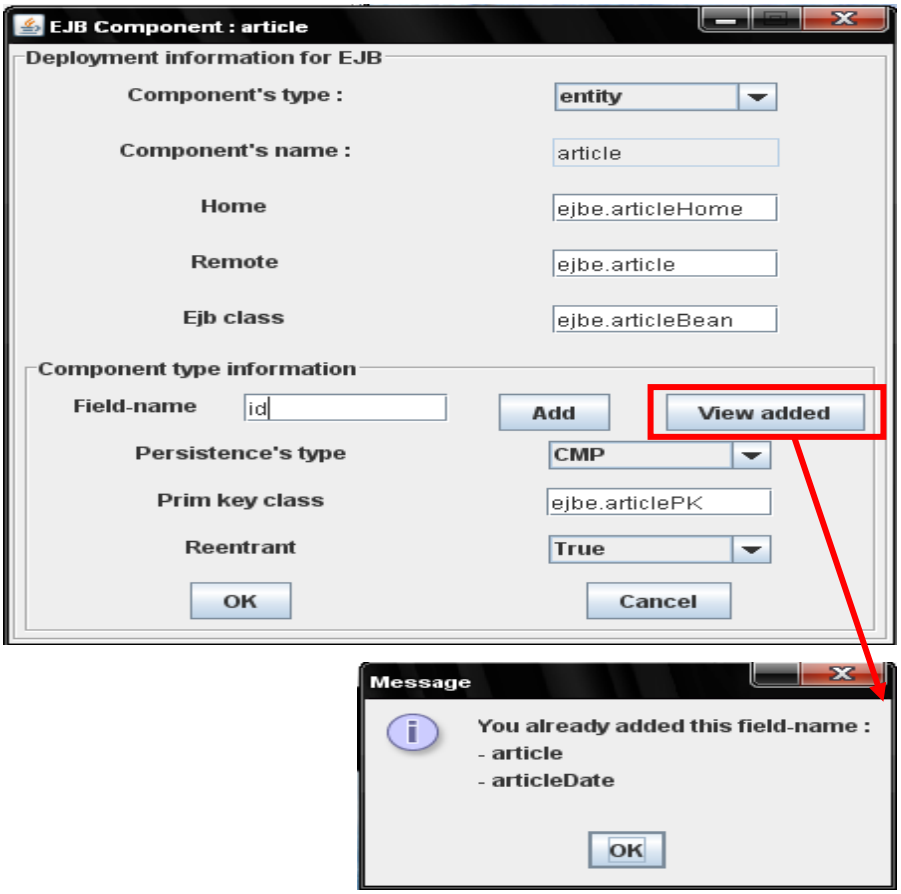


Figure 40 : La fenêtre de saisie des informations d'un composant

La fenêtre de saisie des informations d'un composant est différente pour chaque technologie (les données en entrée nécessaires sont différentes). La figure 40 montre un exemple de cette fenêtre spécifique pour un composant EJB Session.

Certaines informations sont remplies automatiquement via le modèle d'application. D'autres sont saisies par l'utilisateur. Une fois toutes les données saisies et validées, l'utilisateur est renvoyé vers la fenêtre de saisie des informations générales (figure 41). A travers cette interface, il retrouve deux listes (les composants traités et les composants non traités) et le composant pour lequel les informations viennent d'être saisies apparait dans la liste des composants traités.

Ceci permet à l'utilisateur de savoir où il en est dans la création du modèle de descripteur de déploiement. Pour EJB, il faut traiter tous les composants pour que le descripteur soit généré alors que pour CCM un descripteur est généré pour chaque composant.

Lorsque l'utilisateur clique sur le bouton « OK » de l'interface représentée par la figure 40, les classes Java sont instanciées. Pour chaque information récupérée dans la fenêtre, on instancie la classe correspondante. Ainsi on est certain de rester conforme aux spécifications de la technologie cible puisque l'interface elle-même est un metamodèle du descripteur de déploiement de la technologie cible. Ci-dessous un exemple pour la classe « entity » pour son attribut persistence-type :

```
(i) EntityTypeImpl entity = new EntityTypeImpl();
(ii) PersistenceTypeTypeImpl persistence = new PersistenceTypeTypeImpl();
(iii)persistence.setValue(persistenceTypeBox.getSelectedItem().toString());
(iv) entity.setPersistenceType(persistence);
```

La première ligne de code (i) déclare un nouvel objet entity de type EntityTypeImpl. La commande = new EntityTypeImpl() initialise l'objet, dans notre cas il crée un objet vide.

La deuxième ligne de code (ii) a le même effet que la première sauf que c'est pour un objet de type PersistenceTypeTypeImpl.

La troisième ligne de code (iii) utilise la méthode setValue de l'objet persistence qui prend comme valeur d'attribut une chaîne de caractères qui représente le type de persistance d'un composant Entity dans la technologie EJB. Pour récupérer le type de persistance dans la fenêtre, il faut récupérer la valeur sélectionnée dans la liste déroulante, ceci se fait à l'aide de la méthode getSelectedItem() de l'objet persistenceTypeBox qui représente la liste déroulante. Ainsi, l'objet persistence est instancié et il peut être ajouté à l'objet entity à l'aide de sa méthode setPersistenceType qui prend comme attribut un objet de type PersistenceTypeTypeImpl. Une fois tous les composants traités, l'utilisateur clique simplement sur le bouton « Finish » (Figure 39) qui permet de générer le modèle de descripteur de déploiement. Ci-dessous un code généré pour l'exemple de la Figure 40.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "PUBLIC"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
```

```
  <enterprise-beans>
    <entity>
      <ejb-name>article</ejb-name>
      <home>ejbe.articleHome</home>
      <remote>ejbe.article</remote>
      <ejb-class>ejbe.articleBean</ejb-class>
      <persistence-type>CMP</persistence-type>
      <prim-key-class>ejbe.articlePK</prim-key-class>
      <reentrant>True</reentrant>
      <cmp-field>
        <field-name>article</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>articleDate</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>id</field-name>
      </cmp-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

Descripteur de déploiement généré pour EJB

b. La personnalisation du plan de déploiement

Le modèle de plan de déploiement obtenu est au niveau PIM (*Platform independ model*) schématisé par la figure 37 (processus 3/4). Ce modèle de plan de déploiement est composé de placements. Un placement est une association composant-site.

Ce modèle de plan générique doit être personnalisé, c'est-à-dire le transformer vers un modèle de plan de déploiement personnalisé qui sera au niveau PSM (Platform specific model). Pour cela, il sera demandé à l'utilisateur de choisir le modèle de plan générique à personnaliser (Figure 41, Deployment plan name) et la technologie cible vers laquelle le ce modèle sera projeté (Figure 41, Technology target).

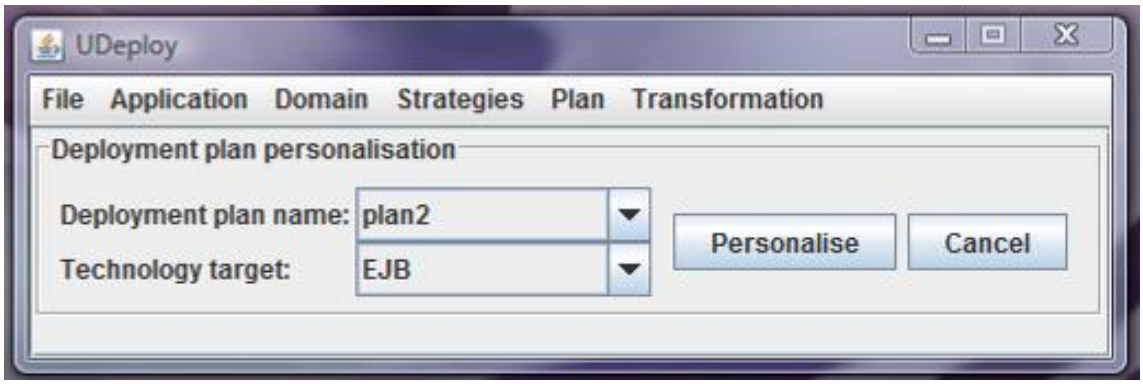


Figure 41 : La personnalisation du modèle de plan de déploiement

6.2.4 L'exécution du plan de déploiement

Normalement, le plan de déploiement personnalisé doit être exécuté par la plateforme cible (SofaNode pour les composants SOFA et StarCCM ou OpenCCM pour les composants CCM). Mais malheureusement, les modèles à composants comme

Fractal, EJB et COM+ ne proposent pas de modèle de plan de déploiement global qui peuvent être exécutés par la suite. Par conséquent, le modèle de plan de déploiement personnalisé pour ces modèles à composants sera exécuté par notre infrastructure. L'exécution de ce plan correspond à la mise en marche des serveurs, au chargement des composants dans les serveurs (via le modèle de plan de déploiement, le site de placement du composant est connu) et à l'établissement des connexions.

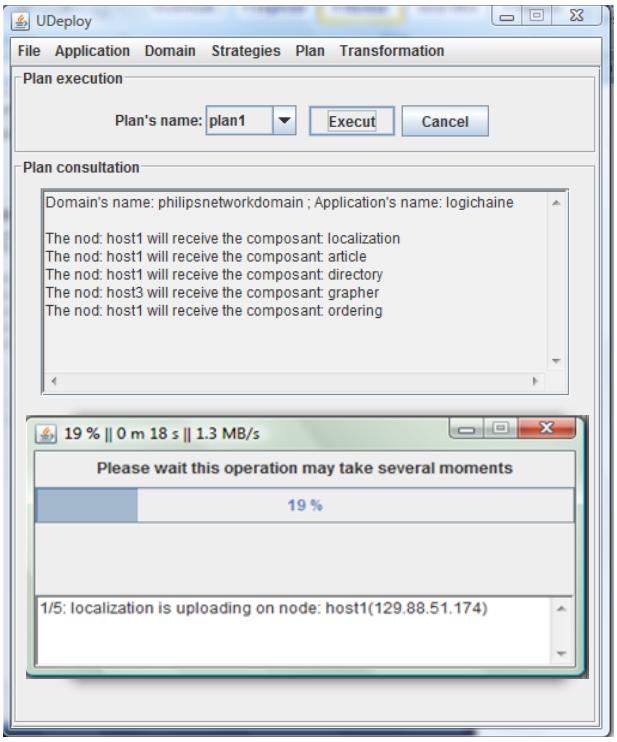


Figure 42 : L'exécution du plan

Lorsque nous exécutons un modèle de plan de déploiement personnalisé, nous commençons par demander à l'utilisateur le nom du plan à exécuter (Figure 32, plan's name). Lorsque l'utilisateur choisit un plan, UDeploy installe le « code de chaque composant » sur le site approprié. Comme le processus d'installation peut durer un certain temps s'il y a beaucoup de composants à déployer, nous avons dû mettre en place une barre d'avancement des taches.

Pour créer une barre des tâches, nous avons commencé par créer une nouvelle fenêtre où nous pourrons mettre l'état d'avancement. Une fois la fenêtre créée, nous calculons la taille totale des répertoires à copier. Cette étape permet à l'utilisateur de connaître l'avancement de la tâche. Lorsque la copie débute, nous devons créer des *threads* qui permettront de vérifier combien d'octets on été copiés sur les serveurs. Si nous connaissons ce nombre d'octets, nous pouvons faire un rapport avec la taille totale qui nous donne le pourcentage d'avancement. Pour limiter le nombre de rafraîchissement et augmenter la performance de la copie, nous pouvons arrêter le *thread* un certain temps avec la méthode *sleep()*. Si nous ajoutons un autre traitement qui stocke les nombres d'octets copiés sur le serveur avant et après le *sleep()*, nous

pouvons connaître la vitesse de copie car nous savons combien de temps le Thread à été endormi.

Dans notre cas, nous avons créé deux *threads*, un pour obtenir le pourcentage d'avancement et l'autre pour la vitesse d'exécution.

```
/**
 * thread pour actualiser la barre de chargement
 *
 */
public class MonRunnable  implements Runnable
{
public void run()
{
        while (!onafinit)
        {
        try{
        monThread.sleep(500);//on stop le Thread pendant 5ms
        progression=StrictMath.round(((nboctetencours*100)/tailletotale));
        if (!(vitesse==0)){
                cadre.setTitle(progression+" % || "+da.getMinutes()
                +" m "+da.getSeconds()+" s || "+vitesse+" MB/s");
        }
        progress.setValue(progression);//on met à jour la barre de tâche
        panneau.paintComponents(panneau.getGraphics());//on met à jour l'affichage de la
         fenêtre
        }catch(Exception e){e.printStackTrace();}
        }
        cadre.dispose();//si on finit de copier alors on ferme la fenêtre}}
/**
 * thread pour actualiser le temps de copie restant
 *
 */
public class MonRunnable2  implements Runnable
{
public void run()
{
        while (!onafinit)
        {
        try{
                long tailletelechargeravantsleep=nboctetencours;
                monThread.sleep(2000);
                long tailletelechargerapressleep=nboctetencours;
                long tailletelechargerpendantsleep=tailletelechargerapressleep-
                tailletelechargeravantsleep;
                vitesse = (float) (tailletelechargerpendantsleep/2);
                vitesse=vitesse/1000000;
                vitesse=vitesse*100;
                vitesse=StrictMath.round(vitesse);
                vitesse=vitesse/100;
                nbsec=StrictMath.round((((tailletotale-
                        nboctetencours)/1000000)/vitesse));
                da=new Date(nbsec*1000);
                if (!(vitesse==0)){
                        cadre.setTitle(progression+" % || "
                                +da.getMinutes()
                                +" m "+da.getSeconds()
                                +" s || "+vitesse+" MB/s");
                }
        }catch(Exception e){e.printStackTrace();}
        }
        cadre.dispose();}}
```

7

UNE ETUDE DE CAS

7.1 LA DESCRIPTION DU CAS D'ETUDE

Nous présentons un cas d'étude pour le déploiement d'une application à base de composants logiciels hétérogènes. Il nous parait intéressant de présenter cette étude pour illustrer d'une part la difficulté de prise en compte du déploiement de telle application dans les intergiciels et d'autre part de montrer les risques d'échec du déploiement dus au manque de flexibilité dans le processus de déploiement.

L'application que nous souhaitons déployer permet de gérer toute la chaine logistique d'une entreprise de vente de matériels informatiques depuis la production des articles jusqu'à la phase de livraison chez le client distributeur. Pour simplifier nous appellerons notre application SCMsoft par la suite inspirée de « my SAP Supply Chain Management ». L'application SCMsoft est constituée de cinq composants qui sont *article, commande, annuaire, localisation* et *graphe*. Chaque composant joue un rôle spécifique :

- « **Article** », est un composant EJB entité. A l'état actuel du système, trois versions du composant ont été développés et sont disponibles dans le base d'application : Article_EJB-v1, Article_EJB-v2 et Article_EJB-v3. Le composant *article* permet d'enregistrer des informations sur les produits fabriqués comme le numéro de série, la fiche technique, la date de production et la position du produit dans l'entrepôt d'usine.
- « **Commande** » existe en trois implémentations qui sont EJB, CCM et SOFA. A l'état actuel du système, trois versions du composant ont été développées pour chaque technologie et sont disponibles dans la base d'application : Commande_EJB-v1, Commande_EJB-v2, Commande_EJB-v3, Commande_CCM-v1, Commande_CCM-v2, Commande_CCM-v3, Commande_SOFA-v1, Commande_SOFA-v2 et Commande_SOFA-v3. Le composant *commande* permet aux distributeurs de produits de passer des commandes au magasin d'usine, c'est un système d'Échange de Données Informatiques (IDE). Chaque commande est identifiée de façon unique par le numéro d'envoi et le code d'immatriculation du client qui a passé la commande.
- « **Annuaire** » est un service web de type page jaune. Une seule implémentation est disponible pour ce service web Annuaire_SOAP_WSDL-v1. Le service *annuaire* utilise le protocole SOAP (*Simple Object Access Protocol*) pour l'échange de message et le WSDL (*Web Services Description Language*) pour la description du service. Le service annuaire est utilisé par le composant *graphe*.
- « **Localisation** » est un composant CCM, qui donne des informations sur la position géographique du colis. Elle utilise un système d'Identification par Radio Fréquence (RFID) qui permet de géo-localiser les colis à temps réels. Trois versions du composant ont été développées et sont disponibles dans la base d'application : Localisation_CCM-v1, Localisation_CCM-v2 et Localisation_CCM-v3.
- « **Graphe** » est un composant composite Sofa, formé de deux composants primitifs « **Figure** » et « **Estimateur** ». Le composant *estimateur* estime le temps probable d'arrivage de la commande en faisant un calcul entre la position géographique du

colis via le composant «localisation» et l'adresse d'arrivage à l'aide du service web « annuaire ». Et, le composant *figure* permet l'affichage de ces informations. Cette fonctionnalité permet à chaque instant, s'il y a besoin de visualiser à temps réel l'état de la commande. A l'état actuel du système, trois versions du composant ont été développés et sont disponibles dans le *repository* application : Graphe_SOFA-v1 (Figure_SOFA-v1, Estimateur_SOFA-v1), Graphe_SOFA-v2 (Figure_SOFA-v2, Estimateur_SOFA-v2) et Graphe_SOFA-v3 (Figure_SOFA-v3, Estimateur_SOFA-v3).

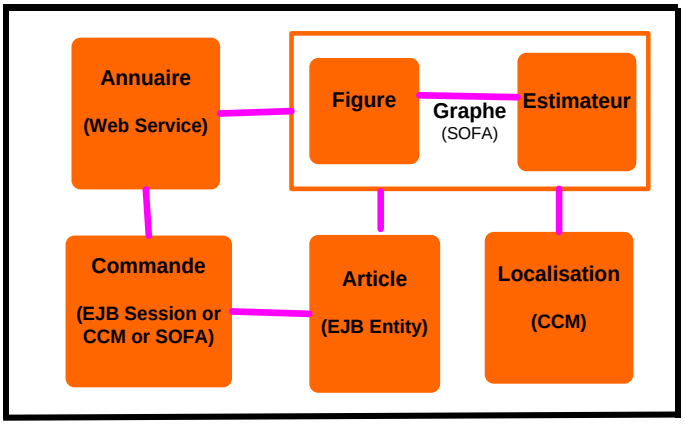


Figure 43 : l'application SCMsoft

Le tableau 4 décrit pour chaque composant, ses différentes implémentations, ses contraintes logicielles et matérielles. Dans ce tableau nous ne présentons pas toutes les implémentations seules les dernières versions sont décrites. Toutes les versions sont disponibles dans la base d'application.

Tableau 4 : Le modèle d'application SCMsoft

COMPOSANT	IMPLEMENTATIONS	CONTRAINTES LOGICIELLES	CONTRAINTES MATERIELLES
COMMANDE	COMMANDE_CCM-v3	OPENORB 1.3.1 JAVA VIRTUAL MACHINE = JRE 1.4.2-02	MASS STORAGE >=2 GO
	COMMANDE _EJB-v3	JBOSS 4.2.3.GA JAVA VIRTUAL MACHINE = JRE 1.4.2-02	MASS STORAGE >=1 GO
	COMMANDE _SOFA-v3	SOFARUNTIME JAVA VIRTUAL MACHINE =JRE 1.4.2-02	MASS STORAGE >=1 GO
ANNUAIRE	ANNUAIRE_SOAP_WSDL-v1 EST DEJA DEPLOYE SUR UN SERVEUR « IBM LOTUS DOMINO »		
GRAPHE	GRAPHE_SOFA-v3	SOFARUNTIME JAVA VIRTUAL MACHINE =JRE 1.4.2-02	MASS STORAGE >=50 M0
ARTICLE	ARTICLE_EJB-v3	JBOSS 4.2.3.GA JAVA VIRTUAL MACHINE = JRE 1.4.2-02 VERSION = ORACLE 8.1.5	RAM > =3062 MO MASS STORAGE >=450 GO
LOCALISATION	LOCALISATION_CCM-v3	OPENORB 1.3.1 JAVA VIRTUAL MACHINE = JRE 1.4.2-02	RAM> = 512 MO MASS STORAGE>= 72 MO

A l'état actuel du système SCMsoft, il existe trois configurations possibles dont deux sont valides. Les principales configurations pour la dernière version du logiciel sont :

- SCMsoft_configuration1 (**COMMANDE_EJB-v3**, **ANNUAIRE_SOAP_WSDL-v1**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**).

- SCMsoft_configuration2 (**COMMANDE_CCM-v3**, **ANNUAIRE_SOAP_WSDL-v1**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**).

- SCMsoft_configuration3 (**COMMANDE_SOFA-v3**, **ANNUAIRE_SOAP_WSDL-v1**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**).

Les configurations SCMsoft_configuration1 et SCMsoft_configuration2 sont valides tandis que la configuration SCMsoft_configuration3 est invalide car le composant **COMMANDE_SOFA-v3** ne peut pas communiquer avec le composant **ARTICLE_EJB-v3**.

Solis est une jeune entreprise informatique qui produit et vend du matériel informatique à trois chaines de distribution dans le monde (France, États-Unis et Espagne) et face à une augmentation de ses commandes et ainsi pour faciliter la gestion de sa chaine logistique, la société Solis à décider d'acquérir le logiciel SCMsoft.

Il souhaite ainsi, déployer l'application SCMsoft sur son domaine SolisDomaine. Le domaine SolisDomaine est constitué des sites principaux suivants :

- Le site H1 fournit un serveur d'application et un serveur de base de données. Il se trouve dans l'usine en Inde.
- Les sites H2, H3, H4 fournissent chacun un serveur d'application. Il se trouve respectivement dans les locaux des chaines de distribution en France, aux États-Unis et en Espagne.
- Le site PDA1, PDA2 sont connectés à internet et offre un système RFID. Le PDA1, PDA2 se trouvent respectivement sur les moyens de transport qui sont le camion et le bateau. Il dispose au total de 4 PDA de type PDA1 et de 3 PDA de type PDA2. Les PDA sont identifiés de PDA1-1 à PDA1-4 pour ceux de type PDA1 et identifiés de PDA2-1 à PDA1-3 pour ceux de type PDA2. La position de chaque colis est connue à chaque instant grâce à ces PDA.
- Le site Sigma est un serveur web.
- Le site Adèle fournit un serveur d'application.

Sur les sites de type PDA1 sont installés le composant Localisation_CCM-v2 et sur le site Sigma est installé Annuaire_SOAP_WSDL-v1. Ces installations ont été faites lors de précédentes activités de déploiement.

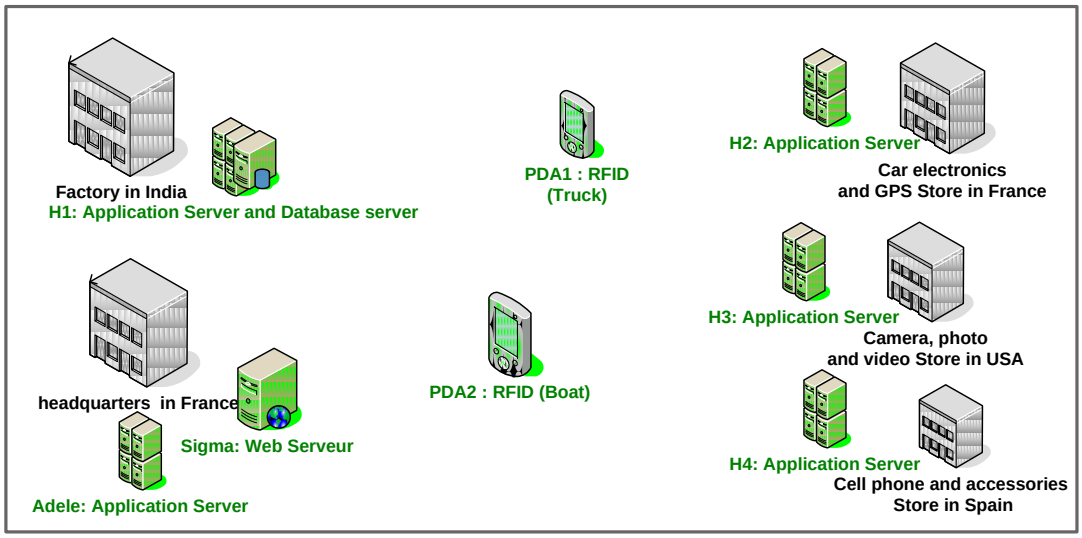


Figure 44 : Le domaine de déploiement (Entreprise Solis)

Le tableau 5 décrit de manière plus détaillée pour chaque site, les ressources logicielles et matérielles dont il dispose.

Tableau 5 : Le modèle de domaine Solis

SITE	RESSOURCES LOGICIELLES	RESSOURCES MATERIELLES
H1: APPLICATION SERVER AND DATABASE SERVER	ORACLE 11G JBOSS 4.2.3.GA JAVA VIRTUAL MACHINE = JRE 1.4.2-02	MASS STORAGE =900 G0 MASS STORAGE =700 GO PROCESSOR = CORE 2 QUAD
H2: APPLICATION SERVER	JBOSS 4.2.3.GA OPENORB 1.3.1 JAVA VIRTUAL MACHINE = JRE 1.4.2-02	RAM= 3062 MO MASS STORAGE= 800 G0
H3: APPLICATION SERVER	JBOSS 4.2.3.GA SOFARUNTIME JAVA VIRTUAL MACHINE = JRE 1.4.2-02	RAM = 3062 MO MASS STORAGE= 800 G0
H4: APPLICATION SERVER	JBOSS 4.2.3.GA JAVA VIRTUAL MACHINE = JRE 1.4.2-02	RAM = 3062 MO MASS STORAGE= 700 G0
PDA1: RFID (TRUCK) (PDA11, PDA12, PDA13, PD14)	OPENORB 1.3.1 JAVA VIRTUAL MACHINE =JRE 1.4.2-02 LOCALIZATION_CCM_v2	RAM = 512 MO MASS STORAGE =4 GO PROCESSOR = CORE 2 DUO
PDA2: RFID (BOAT) (PDA21, PDA22, PDA23)	OPENORB 1.3.1 JAVA VIRTUAL MACHINE =JRE 1.4.2-02	RAM = 512 MO MASS STORAGE =10 GO PROCESSOR = CORE 2 DUO
SIGMA: WEB SERVER	IBM LOTUS DOMINO DIRECTORY_WSDL_v3	MASS STORAGE =160 GO
ADELE: APPLICATION SERVER	SOFARUNTIME JAVA VIRTUAL MACHINE = JRE 1.4.2-02	RAM = 3062 MO MASS STORAGE= 800 G0

7.2 LA MISE EN ŒUVRE DU DEPLOIEMENT

Solis souhaite déployer la configuration SCMsoft_configuration1 (**COMMANDE_EJB-v3**, **ANNUAIRE_SOAP_WSDL-v1**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**) sur le domaine SolisDomaine.

Nous savons que le composant **ANNUAIRE_SOAP_WSDL-v1** est déjà déployé. Il nous reste donc à déployer que les quatre autres composants **COMMANDE_EJB-v3**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**.

L'entreprise Solis souhaite déployer les quatre composants avec les outils de déploiement J2EE. Malheureusement, seuls les composants EJB peuvent être déployés par les outils J2EE alors il décide de déployer les deux composants EJB (**COMMANDE_EJB-v3**, **ARTICLE_EJB-v3**). Mais aussitôt, il se rend compte que l'étape de planification n'est pas prise en compte dans ces outils et que la création du plan de déploiement et du descripteur de déploiement se fait à la main. L'entreprise tente alors une seconde fois de déployer les quatre composants avec les outils de déploiement .NET. Aucun composant ne peut être déployé par les outils .NET car ils ne supportent que le déploiement de composant .NET (COM, COM+).

L'entreprise tente alors une troisième fois de déployer les quatre composants avec D&C/DANCE. Avec D&C le plan de déploiement pourra être calculé mais seule l'exécution du plan de déploiement pour les composants CCM sera effective car la seule implémentation de la spécification D&C est DANCE qui supporte uniquement le déploiement de composant CCM.

L'entreprise n'abandonne pas, il décide alors de déployer les quatre composants dans une infrastructure SOFA. Mais malheureusement, le déploiement dans SOFA n'est pas distribué. Le plan de déploiement sera directement exécuté dans un serveur central en supposant que les sites distants pourront instancier les composants à travers ce serveur (SOFAnode). Un avantage majeur de ce déploiement est que SOFA offre un support DCUP qui perme de gérer l'adaptation dynamique des composants, c'est-à-dire de pouvoir mettre à jour les composants quand ils sont en cours d'exécution. Mais un inconvénient majeur de ce type de déploiement est que tous les composants sont sur un même site, d'une part l'accès au composant peut être long si le serveur SOFAnode est très distant du serveur Dock d'instanciation du composant et d'autre part dès que le site SOFAnode tombe en panne tout le système reste indisponible.

L'entreprise Solis décide alors de déployer les composants **COMMANDE_EJB-v3** et **ARTICLE_EJB-v3** avec l'outil de déploiement J2EE JBOSS ; le composant **LOCALISATION_CCM-v3** avec l'outil DANCE et le composant **GRAPHE_SOFA-v3** avec l'infrastructure SOFA.

L'outil UDeploy permet de déployer tous les composants au travers d'une seule infrastructure au lieu de faire du déploiement unitaire. UDeploy propose donc un cadre unifié qui permet (1) de supporter le déploiement de tous les composants, (2) de définir les stratégies de déploiement propres aux différentes technologies et à l'entreprise Solis et (3) de calculer et de personnaliser les plans de déploiement vers les plates-formes

cibles. Dans ce qui suit nous décrivons comment UDeploy permet de déployer l'application SCMsoft.

7.3 LE DEPLOIEMENT DU LOGICIEL SCMSOFT AVEC UDEPLOY

Nous partons du postulat que l'outil UDeploy est installé sur la machine de l'administrateur système de l'entreprise Solis. Et qu'à partir de cette machine tous les autres sites du domaine Solis sont disponibles. Les *repository* d'application, de domaine, de stratégies et de plan existent et sont accessibles via un url, un login et un mot de passe.

7.3.1 Le modèle d'application

L'administrateur système de l'entreprise Solis commence tout d'abord par modéliser la configuration à déployer. Pour cela, il crée le modèle d'application SCMsoft_configuration1, lui associe les composants (**COMMANDE_EJB-v3**, **GRAPHE_SOFA-v3**, **ARTICLE_EJB-v3**, **LOCALISATION_CCM-v3**) et donne des méta-informations de description pour chaque implémentation ainsi que les contraintes logicielles et matérielles qu'ils requièrent.

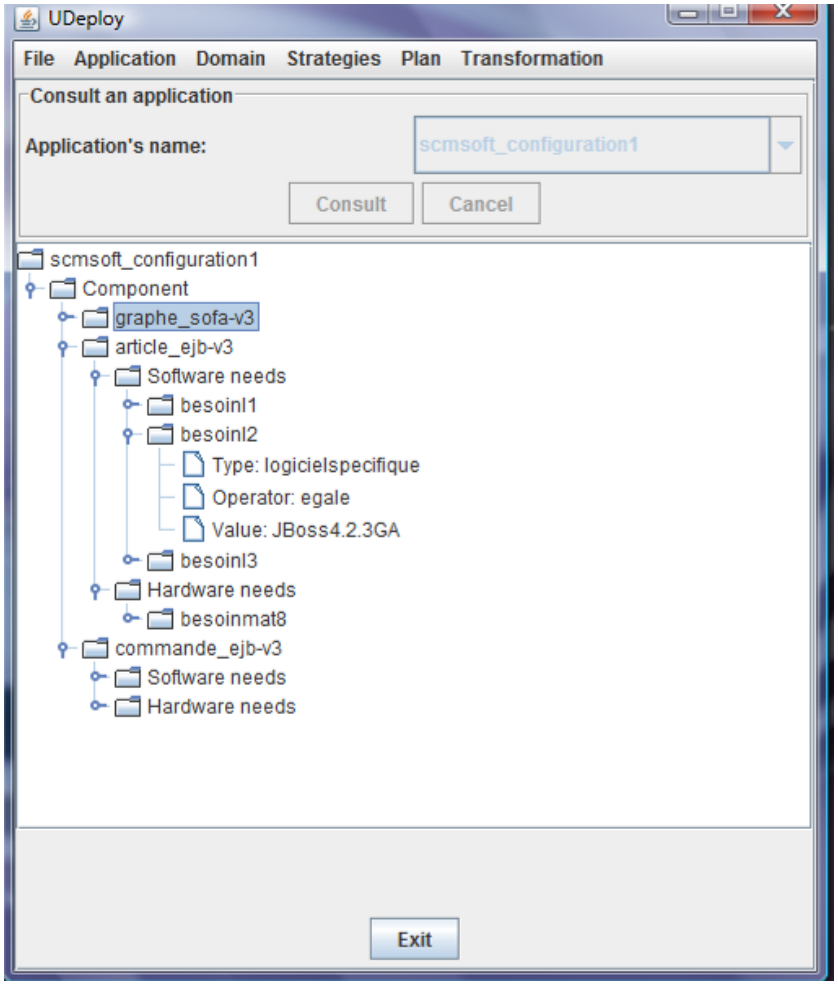


Figure 45 : Le modèle d'application SCMSoft

7.3.2 Le modèle de domaine

Après avoir modélisé l'application, l'administrateur système modélise le domaine de déploiement. Pour cela, il crée un domaine SolisDomaine et lui associe les sites (H1, H2, H3, H4, PDA1-1, PDA1-2, PDA1-3, PDA1-4, PDA2-1, PDA2-2, PDA2-3, SIGMA, ADELE) et donne des méta-informations de description pour chaque site ainsi que les ressources logicielles et matérielles qu'ils offrent.

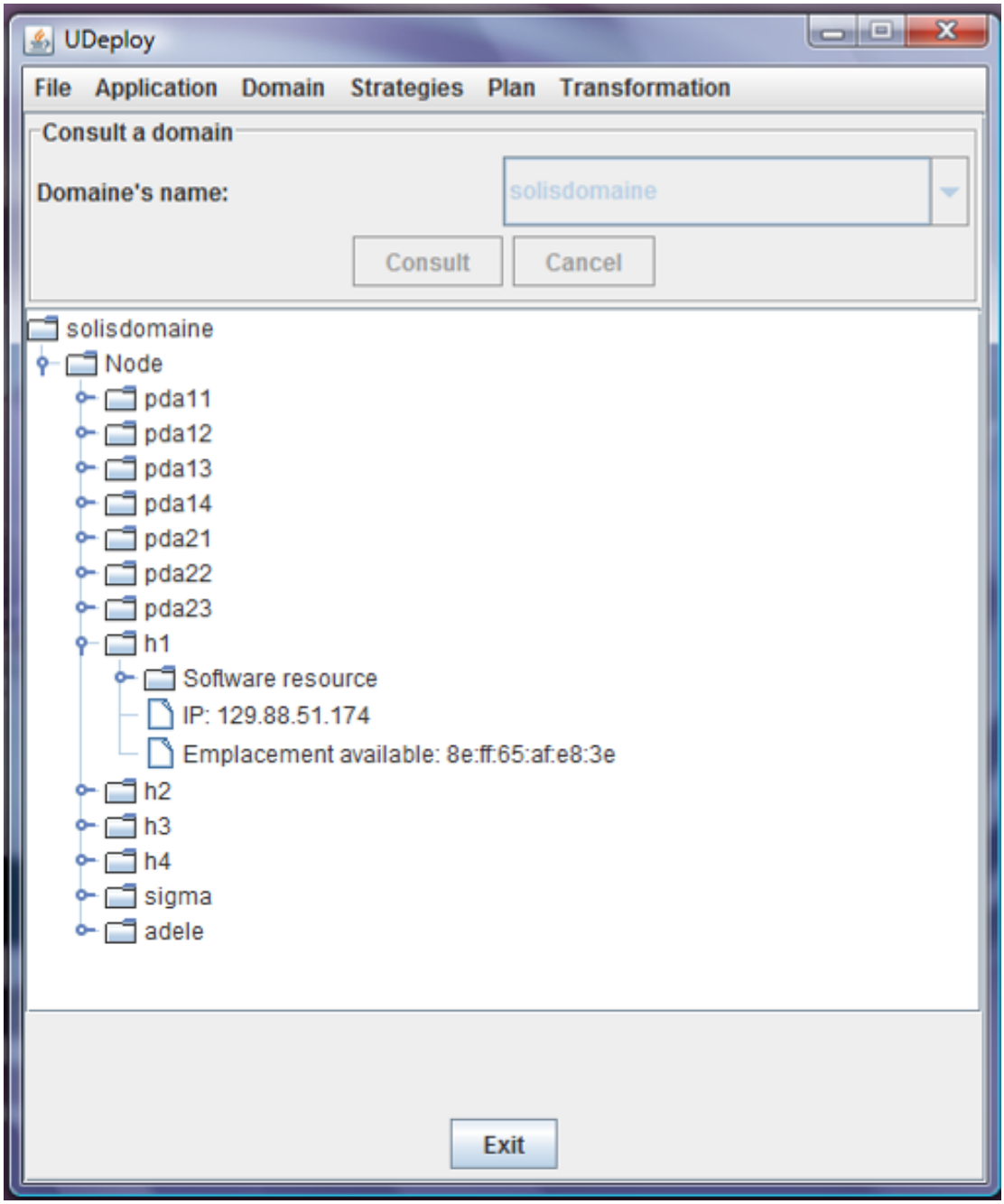


Figure 46 : Le modèle de domaine SolisDomaine

7.3.3 Les stratégies de déploiement

L'administrateur système définit un modèle de stratégies afin de rendre le déploiement flexible. Pour cela, il crée le modèle de stratégies et lui associe les règles (Rule1, Rule2, Rule3 et Rule4) et définit les événements, les conditions et les actions qui leur sont associés.

L'annexe 3 décrit le modèle de stratégies de déploiement contenant les règles 1 et 2. La règle Rule1 exprime que tout composant EJB Entity, EJB Session ou EJB Message-Driven doit être installé sur un site fournissant obligatoirement un serveur d'application. La règle Rule2 exprime que pour tout composant EJB Entity, EJB Session ou EJB Message-driven, le serveur d'application sera par défaut JBOSS4.2.3.GA. Cela veut dire que s'il y a un choix à faire entre plusieurs sites candidats, ceux vérifiant cette règle par défaut seront privilégiés.

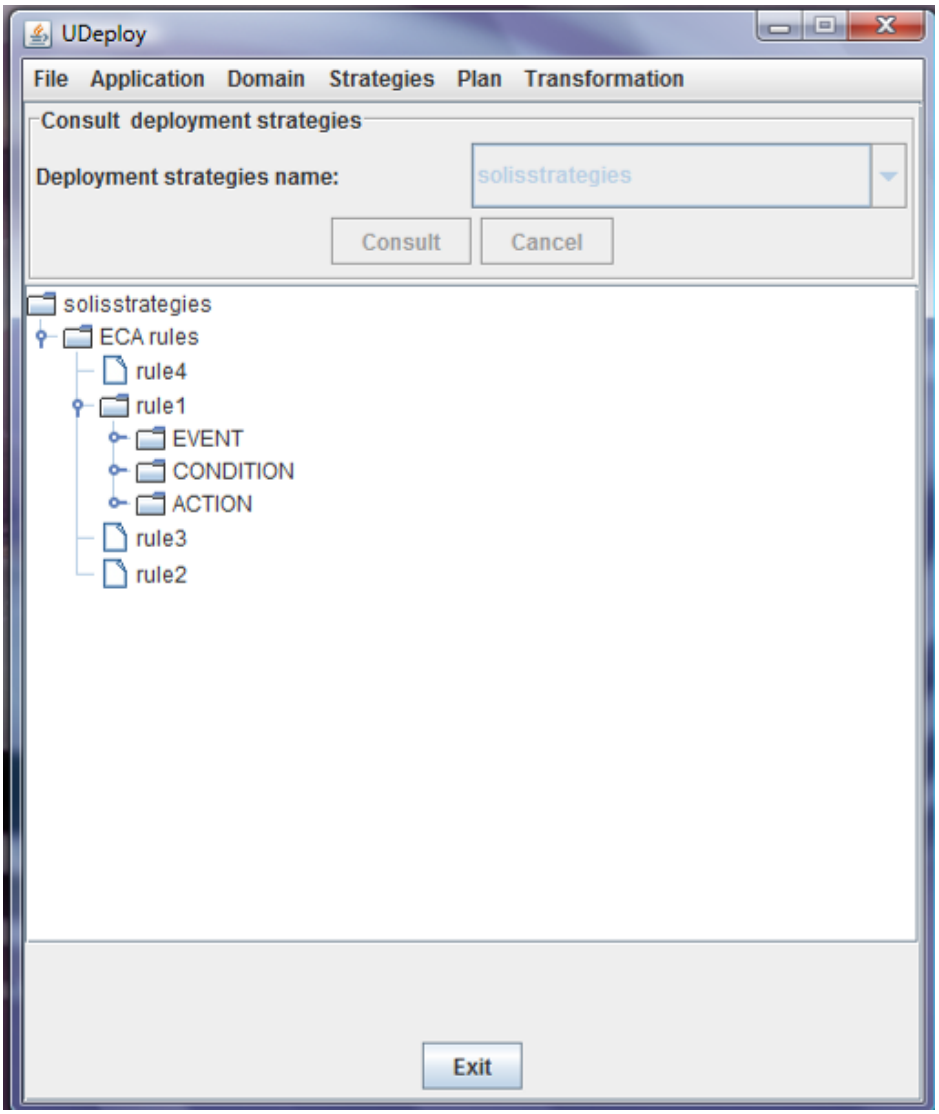


Figure 47 : Le modèle de stratégies SolisStrategies

7.3.4 Le modèle de plan de déploiement

Pour calculer le modèle de plan de déploiement :

1) L'administrateur fournira un modèle d'application **SCMsoft_configuration1**, un modèle de domaine **SolisDomaine** et un modèle stratégies de déploiement **SolisStrategies** pour lesquels il souhaite calculer un modèle de plan de déploiement (Figure 50, processus 1/2).
2) l'administrateur fixera les contraintes matérielles et logicielles des composants de l'application qui doivent être évaluées (Figure 50, processus 2/4).
3) Les placements valides entre les composants et les nœuds du domaine sont calculés suivant l'algorithme de planification (cet algorithme est décrit dans la section 6.2).
4) L'administrateur choisira parmi les placements proposés. Lorsqu'il choisit un placement, nous devons ré-effectuer l'étape 3 car les ressources logicielles et matérielles du nœud choisi auront diminuées (Figure 51, processus 3/4).
5) Une fois, le modèle de plan de déploiement construit avec tous les placements valides, l'administrateur donnera un nom au modèle de plan de déploiement calculé (Figure 52, processus 4/4).

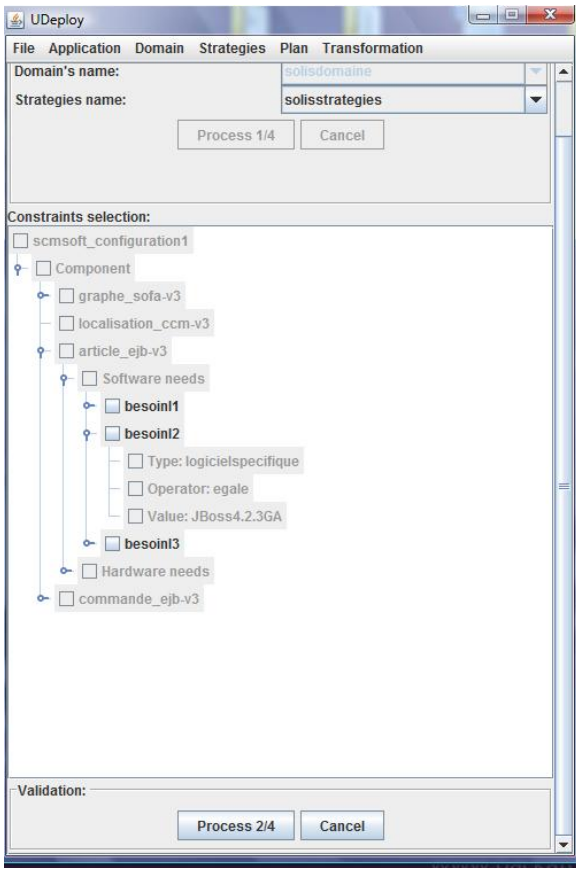


Figure 48 : Le calcul du plan de déploiement (processus 1/4 et 2/4)

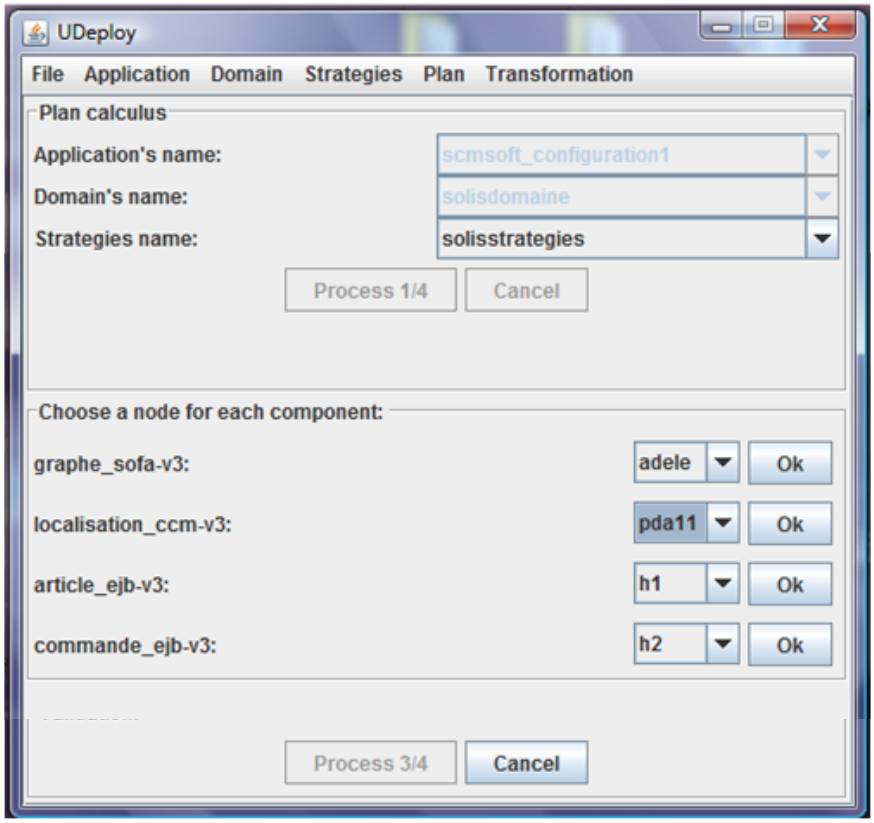


Figure 49 : Le calcul du plan de déploiement (processus 3/4)

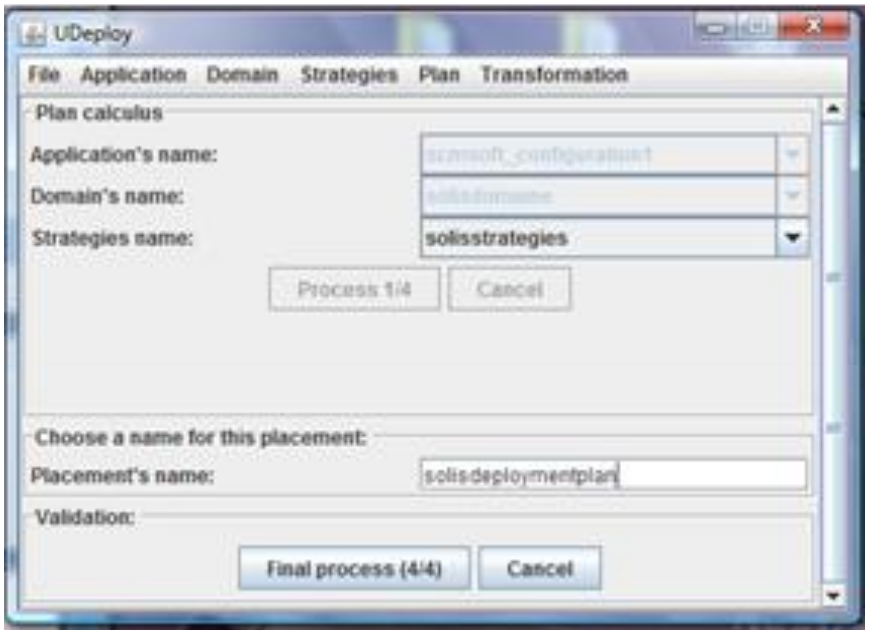


Figure 50 : Le calcul du plan de déploiement (processus 4/4)

Une fois que l'activité de planification est terminée un plan de déploiement générique est produit : *Solisdeploymentplan*. Le modèle de plan complet de *Solisdeploymentplan* est fourni en annexe 3.

Solisdeploymentplan.xml

<DeploymentPlan>
  <name>**Solisdeploymentplan**</name>
  <applicaton>**logichaine**</application>
  <domaine>**SolisDomaine**</domain>
  <strategies>**SolisStrategies**</strategies>
  <placements>
  <placement>
  <component> **Commande_EJB-v3.jar**</component>
  <node> **H2**</node>
  </placement>
  ...
  </placements>
</ DeploymentPlan>

7.3.5 La transformation

a. La génération du descripteur de déploiement

(1) Pour les composants Commande_EJB-v3 et Article_EJB-v3. Les descripteurs suivants sont produits :

Le descripteur de déploiement pour Commande_EJB-v3
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE ejb-jar PUBLIC "PUBLIC" "http://java.sun.com/dtd/ejb-jar_2_0.dtd"> <ejb-jar>   <enterprise-beans>   <session>   <ejb-name>commande</ejb-name>   <home>ejb.commandeHome</home>   <remote>ejb.commande</remote>   <ejb-class>ejb.commandeBean</ejb-class>   <cmp-field><field-name>id</field-name></cmp-field>   <cmp-field><field-name>storenumber</field-name></cmp-field>   <cmp-field>   <field-name>commandeDate</field-name>   </cmp-field>   </session>   </enterprise-beans

Le descripteur de déploiement pour Article_EJB-v3

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE ejb-jar PUBLIC "PUBLIC"
 "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
    <entity>
            <ejb-name>article</ejb-name>
            <home>ejb.articleHome</home>
            <remote>ejb.article</remote>
                <ejb-class>ejb.articleBean</ejb-class>
                <persistence-type>BMP</persistence-type>
                <prim-key-class>ejb.articlePK</prim-key-class>
            <reentrant>True</reentrant>
            <cmp-field>
                    <field-name>serialnumber</field-name>
            </cmp-field>
            <cmp-field>
                    <field-name>ProductionDate</field-name>
            </cmp-field>
            <cmp-field>
                    <field-name>Model</field-name>
            </cmp-field>
    </entity>
    </enterprise-beans
```

(2) Pour le composant CCM Localisation_CCM-v3, le descripteur suivant est produit

Descripteur de déploiement pour Localisation_CCM-v3

```
<corbacomponent>
    <corbaversion>3.0</corbaversion>
    <componentrepid repid="IDL:localisation_ccm-v3:1.0"/>
    <homerepid repid="IDL: localisation_ccm-v3Home:1.0"/>

    <componentkind>
       <session>
           <servant lifetime="container"/>
       </session>
    </componentkind>
    <homefeatures name=" localisation_ccm-v3Home" repid="IDL: localisation_ccm-
        v3Home:1.0">
     </homefeatures>
```

```
    <componentfeatures  name="  localisation_ccm-v3"  repid="IDL:  localisation_ccm-
        v3:1.0">
        <ports>
            <consumes  consumesname="notifier_in" eventtype="IDL:
                localisationName:1.0">
                <eventpolicy policy="normal"/>
            </consumes>
            <uses usesname="quoter_info_in" repid="IDL: localisationQuoter:1.0">
            </uses>
        </ports>
    </componentfeatures>

    <interface name=" localisationQuoter" repid="IDL:localisationQuoter:1.0">
    </interface>
</corbacomponent>
```

(3) Pour le composant composite SOFA Graphe-SOFA-v3, aucun descripteur de déploiement ne sera généré.

b. La personnalisation du plan de déploiement

En personnalisant le plan de déploiement, nous obtenons les plans suivants pour chaque technologie spécifique.

(1) Intergiciel EJB/ JBOSS

Le plan de déploiement personnalisé pour article_EJB-v3 et commande_EJB-v3

```
DeploymentPlanModel EJBDP is

   DeploymentSubPlanModel EJBDSP1 On Node H1 is
     twiddle invoke "jboss.system:service= MainDeployer" deploy file:article-EJB-
v3.jar

   DeploymentSubPlanModel EJBDSP2 On Node H2 is
    twiddle invoke "jboss.system:service= MainDeployer" deploy file:commande-EJB-
v3.jar

DeploymentSubPlanModel EJBDSP3 On Node H3 is
    twiddle invoke "jboss.system:service= MainDeployer" deploy file:commande-EJB-
v3.jar
DeploymentSubPlanModel EJBDSP4 On Node H4 is
    twiddle invoke "jboss.system:service= MainDeployer" deploy file: commande-EJB-
v3.jar
```

(2) Intergiciel CCM/(StarCCM ou MicoCCM)

Le plan de déploiement personnalisé pour Localisation_CCM-v3

```
DeploymentPlanModel CCMDP is
   DeploymentSubPlanModel CCMDSP1 On Node PDA11 is
       Remove (Localisation_CCM-v2)

       Install (Localisation_CCM-v3)

   DeploymentSubPlanModel CCMDSP2 On Node PDA12 is
```

```
        Remove (Localisation_CCM-v2)

        Install (Localisation_CCM-v3)

    DeploymentSubPlanModel CCMDSP3 On Node PDA13 is
        Remove (Localisation_CCM-v2)

        Install (Localisation_CCM-v3)

    DeploymentSubPlanModel CCMDSP4 On Node PDA14 is
        Remove (Localisation_CCM-v2)

        Install (Localisation_CCM-v3)

    DeploymentSubPlanModel CCMDSP5 On Node PDA21 is
        Install (Localisation_CCM-v3)

    DeploymentSubPlanModel CCMDSP6 On Node PDA22 is
        Install (Localisation_CCM-v3)

    DeploymentSubPlanModel CCMDSP7 On Node PDA23 is
        Install (Localisation_CCM-v3)
```

(3) Intergiciel SOFA (runtime SOFAnode)

Le plan de déploiement personnalisé pour Graphe_SOFA-v3

```
<?xml version="1.0" encoding="UTF-8"?>
<depl-plan name="SOFADP" component ='Graphe-sofa-v3" sofanode="Adele">

   <depl-subc name="figure-SOFA-v3.jar"/>
   <depl-subc name="estimateur-SOFA-v3.jar"/>

</depl-plan>
```

7.3.6 L'exécution des modèles de plan de déploiement

L'intergiciel SOFA exécutera directement le modèle de plan de déploiement SOFADP.

Les scripts EJBDSP1 à EJBDSP4 et les scripts CCMDSP1 à CCMDSP7 seront exécutés par l'outil UDeploy.

7.4 L'EVALUATION TECHNIQUE

Plusieurs méthodes existent pour l'évaluation des logiciels libres. Les cinq principales méthodes sont :

- *Open Source Maturity Model (OSMM)* de Capgemini
- *Open Source Maturity Model (OSMM)* de Navica
- *Methodology of Qualification and Selection of Open Source software (QSOS)* de Atos Origin,
- *Open Business Readiness Rating (OpenBRR)* de Carnegie Mellon- Silicon Valley
- *QualiPSo OpenSource Maturity Model (OMM)* de QualiPSo project,

Nous avons opté pour la méthode d'évaluation OSMM [Capgemini, 2010] pour l'évaluation industrielle de l'outil UDeploy. OSMM permet d'évaluer une réalisation logicielle sur quatre dimensions (tableau 6) importantes qui sont : le produit, l'intégration, l'utilisation et l'adoption.

Tableau 6 : OSSM Évaluation

Open Source Maturity Model product evaluation form			
******** PRODUCT INDICATORS ********			
Product Information			
Product name			
Homepage product			
Version			Release date:
Evaluator Information	**Name**		**Company**
First Product Evaluator (FPE)			
Second Product Evaluator (SPE)			
Product indicators	**Score (FPE)**	**Score (SPE)**	**Motivation**
Age			
Licensing			
Human hierarchies			
Selling points			
Developer community			
Modularity			
Collaboration with other products			
Standards			
Support			
Ease of deployment			
User community			
Market penetration			

Le produit, sur ce point l'outil UDeploy a 3 points sur 6, cela étant dû à :

- Un manque de grande communauté de développeurs (petite communauté)
- Un manque de référentiel CVS qui permettra d'y apporter des contributions.
- Les aspects vendeurs sont non traités.

L'intégration et l'utilisation récoltent des points de 6 sur 6, cela est dû :

- à la modularité de l'outil ;
- à la facilité d'interopérabilité avec d'autres systèmes ;
- aux standards de développement (Java, xml) ;
- à la facilité de déploiement.

L'adoption récolte 2 points sur 6, cela est surtout dû au fait que pendant la phase de spécification de l'outil UDeploy nous n'avons pas pris contact avec une large communauté d'utilisateurs d'outils de déploiement.

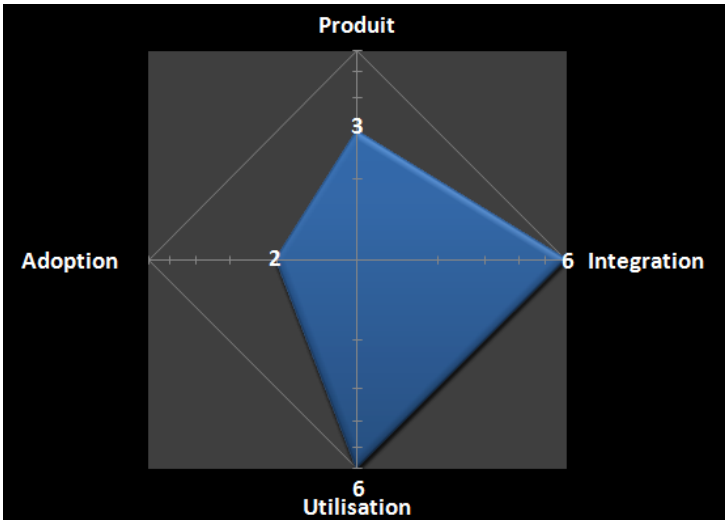


Figure 51 : L'évaluation de UDeploy par la méthode OSMM

8

LA CONCLUSION ET LES PERSPECTIVES

8.1 LES CONTRIBUTIONS

Le déploiement est un processus complexe particulièrement quand il s'agit de déployer un large système sur une grande infrastructure matérielle.

Les solutions existantes pour déployer les systèmes à base de composants sont généralement développées de manière ad' hoc, c'est-à-dire qu'elles sont spécifiques à des technologies.

Ces dernières années, il y a eu beaucoup de travaux académiques focalisés sur une nouvelle génération de systèmes. Ces approches améliorent la transition technologique. Ils ont montré la potentialité d'utiliser une approche dirigée par les modèles telle que MDA. Les modèles définis sont fondés sur des abstractions expressives et simples comme l'application, le domaine et le processus de déploiement. Ce travail est une proposition à cette nouvelle génération de systèmes. Les contributions sont comme suit :

- Nous avons introduit les concepts clés du domaine d'étude qui sont le déploiement et l'approche à composant.
- Nous avons fait une étude de l'état de l'art et de la pratique qui nous a permis d'avoir une vision globale des problématiques de déploiement. L'étude comparative des différentes approches montrent que les approches sont similaires dans leur objectif mais différentes dans leur mode d'implémentation. Ainsi, le bilan tiré de l'état de l'art et de la pratique nous a permis d'arriver à la proposition d'un noyau commun aux différentes approches.
- Nous proposons un cadre conceptuel de déploiement. Nous présentons premièrement l'architecture globale de notre environnement de déploiement et deuxièmement nous présentons les différents éléments constituants ce système environnement.
- Nous proposons un moteur de planification du déploiement. Ce moteur (1) prend en entrée des méta-informations relatives à l'application, à l'infrastructure et aux stratégies de déploiement, (2) il génère un plan de déploiement générique et (3) un descripteur de déploiement qui seront par la suite personnalisés.
- Nous proposons un langage de transformation de modèle. Ce langage permet le passage d'un modèle spécifique vers un modèle indépendant ou le passage d'un modèle indépendant vers un modèle spécifique.
- Nous avons proposé un prototype UDeploy afin de valider l'approche et de montrer sa faisabilité sur des intergiciels cibles. Une étude de cas qui illustre l'utilisation de l'outil a été proposée.

Nous espérons ainsi contribuer au développement d'une nouvelle génération d'environnements de déploiement dirigés par les modèles.

8.2 LES PERSPECTIVES

Les perspectives à court terme

- La consolidation de l'outil UDeploy en menant des expérimentations plus larges sur d'autres approches à composants académiques et industrielles.
- L'amélioration du méta-modèle de stratégies en offrant plus de sémantique pour l'expression des politiques de déploiement.

Les perspectives à long terme

- L'ajout d'une dimension importante du déploiement qui n'est pas prise en compte dans ce travail et qui est la réalisation de protocole d'exécution de plan plus élaboré. En effet, nous avons identifié trois types d'exécution de plan (initiale, ultérieure statique et ultérieure dynamique). L'exécution initiale faisant référence au premier déploiement des composants d'une application. L'exécution ultérieure, appelé aussi reconfiguration, son but est de permettre à un système déjà déployé d'évoluer. Elle peut être statique (quand l'application est désactivée) ou dynamique (quand l'application est en cours d'exécution). Seules l'exécution initiale et l'exécution ultérieure statique sont prises en compte dans ce travail. Nous souhaitons exploiter pour cela, les travaux réalisés dans DYVA [Ketfi et al., 2002] [Ketfi and Belkhatir, 2004], dans CADEComp [Ayed et al., 2008] [Ayed et al., 2005], dans Delegation [Yang et al., 2009], dans Chisel [Keeney and Cahill, 2003] et d'autres travaux qui se font dans le même domaine comme [Villemur and Hammami, 2008] [Segarra and André, 2009] [Rodriguez et al., 2008]. Nous comptons capitaliser les apports de ces travaux.
- L'amélioration de la transformation de modèle, fasse à une complexité technique nous avons proposé un langage de transformation mixte basé sur le QVT ATL et sur des algorithmes de transformation. Les algorithmes de transformation permettent de traiter les composants technologiques EJB, CCM, SOFA et Fractal. Ils sont non adaptables à d'autres approches à composants. Un travail important doit être fait à ce niveau, c'est-à-dire proposer un algorithme de transformation générique qui permettra que d'autres approches technologiques puissent être traitées par UDeploy avec peu d'effort.
- L'extension de l'environnement pour le déploiement des services web [Zeng et al., 2004] [Yu et al., 2008] [Maamar et al., 2009] [Gergic et al., 2002] [Fong et al., 2009] [Pahl, 2005]. Pour y arriver nous devons effectuer des études importantes sur la description des services web, le protocole d'interconnexion des services web, le protocole d'interaction des services web avec des composants logiciels technologiques et la création d'application par souscription à des services web.

9
LA BIBLIOGRAPHIE

[Ahmad and Basson, 2009] Ahmad, A. and Basson, H. (2009). Software evolution modelling: an approach for change impact analysis. In *Proceedings of the 6th International Conference on Frontiers of Information Technology*, FIT '09, pages 56:1–56:4, New York, NY, USA. ACM.

[Alliance, 2005] Alliance, O. (2005). OSGi 4.0 release. Specification available at http://www.osgi.org/.

[ApacheAnt, 2009] ApacheAnt (2009). The apache ant project. Available at http://ant.apache.org/.

[ApacheMaven, 2009] ApacheMaven (2009). The apache maven project. Available at http://maven.apache.org/.

[Ayed et al., 2005] Ayed, D., Belhanafi, N., Taconet, C., and Bernard, G. (2005). Deployment of component-based applications on top of a context-aware middleware. In *IASTED Conf. on Software Engineering*, pages 414–419.

[Ayed et al., 2008] Ayed, D., Taconet, C., Bernard, G., and Berbers, Y. (2008). Cadecomp: Context-aware deployment of component-based applications. *J. Netw. Comput. Appl.*, 31(3):224–257.

[Baker et al., 1995] Baker, M., Fox, G., Yau, H., Baker, M. A., Fox, G. C., and Yau, H. W. (1995). Cluster computing review. Technical report.

[Bastien and Scapin, 1993] Bastien, J. M. C. and Scapin, D. L. (1993). Preliminary findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces. In *INTERCHI Adjunct Proceedings*, pages 187–188.

[Bastien et al., 1996] Bastien, J. M. C., Scapin, D. L., and Leulier, C. (1996). Looking for usability problems with the ergonomic criteria and with the iso 9241-10 dialogue principles. In *CHI Conference Companion*, pages 77–78.

[Bastien et al., 1999] Bastien, J. M. C., Scapin, D. L., and Leulier, C. (1999). The ergonomic criteria and the iso/dis 9241-10 dialogue principles: a pilot comparison in an evaluation task. volume 11, pages 299–322.

[Berman et al., 2003] Berman, F., Fox, G., and Hey, A. J. G. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc., New York, NY, USA.

[Bézivin et al., 2006] Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., and Lindow, A. (2006). Model transformations? transformation models! In *MoDELS*, pages 440–453.

[Bures et al., 2006] Bures, T., Hnetynka, P., and Plasil, F. (2006). Sofa 2.0: Balancing advanced features in a hierarchical component model. In *SERA*, pages 40–48. IEEE Computer Society.

[Buyya, 1999] Buyya, R. (1999). *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Capgemini, 2010] Capgemini (2010). Osmm (open source maturity model). Specification available at http://osspartner.com/portail/.

[Carzaniga, 1997] Carzaniga, A. (1997). A characterization of the software deployment process and a survey of related technologies. Technical Report 97-84, Dipartimento di Elettronica e Informazione, Politecnico di Milano.

[Carzaniga et al., 1998] Carzaniga, A., Fuggetta, A., Hall, R. S., van der Hoek, A., Heimbigner, D., and Wolf, A. L. (1998). A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Department of Computer Science, University of Colorado.

[Condor, 2009] Condor (2009). 7.4.1 release. Specification available at http://www.cs.wisc.edu/condor/.

[Councill et al., 2000] Councill, B., Flynt, J. S., Mehta, A., Speed, J. R., and Shaw, M. (2000). Component-based software engineering and the issue of trust. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 661–664, New York, NY, USA. ACM. Chairman-Heineman, George T.

[Cunin et al., 2005] Cunin, P.-Y., Merle, N., and Lestideau, V. (2005). ORYA: A strategy oriented deployment framework. In *Component Deployment*, pages 177–180. Springer.

[Deng et al., 2005] Deng, G., Balasubramanian, J., Otte, W., Schmidt, D. C., and Gokhale, A. S. (2005). Dance, a qos-enabled component deployment and configuration engine. In *Component Deployment*, pages 67–82.

[Devanbu et al., 1990] Devanbu, P. T., Brachman, R. J., Selfridge, P. G., and Ballard, B. W. (1990). Lassie—a knowledge-based software information system. In *ICSE '90: Proceedings of the 12th international conference on Software engineering*, pages 249–261, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Dibo and Belkhatir, 2009] Dibo, M. and Belkhatir, N. (2009). Challenges and perspectives in the deployment of distributed components-based software. In *ICEIS(3)*, pages 403–406.

[Dibo and Belkhatir, 2010a] Dibo, M. and Belkhatir, N. (2010a). Defining an unified meta modeling architecture for deployment of distributed components-based software applications. In *ICEIS*, Funchal, Madeira, Portugal. 12th International Conference on Enterprise Information Systems (ICEIS).

[Dibo and Belkhatir, 2010b] Dibo, M. and Belkhatir, N. (2010b). A generic framework for distributed component-based software systems deployment: case study and tool description. In *ENASE*, Athens, Greece. 5th International Conference on Evaluation of Novel Approaches to software Engineering (ENASE).

[Dibo and Belkhatir, 2010c] Dibo, M. and Belkhatir, N. (2010c). Model-driven deployment of distributed components-based software. In *ICSOFT*, Athens, Greece. 5th International Conference on Software and Data Technologies(ICSOFT).

[Dibo and Belkhatir, 2010d] Dibo, M. and Belkhatir, N. (2010d). Méta-modèles dédiés au déploiement de logiciels à base de composants distribués. *Revue I3*.

[Dittrich et al., 1995] Dittrich, K. R., Gatziu, S., and (eds.), A. G. (1995). The active database management system manifesto: A rulebase of adbms features. pages 3–20. Springer.

[Dix et al., 2003] Dix, J., Muñoz-Avila, H., Nau, D. S., and Zhang, L. (2003). Impacting shop: Putting an ai planner into a multi-agent environment. *Ann. Math. Artif. Intell.*, 37(4):381–407.

[Dochez, 2009] Dochez, J. (2009). Jsr 88: Java enterprise edition 5 deployment api specification. Available at http://jcp.org/aboutJava/communityprocess/mrel/jsr088/index.html.

[Dolstra et al., 2004a] Dolstra, E., Jonge, M. D., and Visser, E. (2004a). Nix: A safe and policy-free system for software deployment. In *18th Large Installation System Administration Conference (LISA '04*, pages 79–92. USENIX.

[Dolstra et al., 2004b] Dolstra, E., Visser, E., and de Jonge, M. (2004b). Imposing a memory management discipline on software deployment. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 583–592, Washington, DC, USA. IEEE Computer Society.

[Dufrêne and Seinturier, 2008] Dufrêne, G. and Seinturier, L. (2008). Un adl pour les architectures distribuées à composants hétérogènes. In *CAL*, pages 79–90.

[Engine, 2009] Engine, S. G. (2009). 6.2 release. Specification available at http://www.sun.com/software/sge/.

[Fichman and Kemerer, 1997] Fichman, R. G. and Kemerer, C. F. (1997). Object technology and reuse: Lessons from early adopters. *Computer*, 30(10):47–59.

[Flexera, 2010] Flexera (2010). Installshield. Available at http://www.flexerasoftware.com/products/installshield.htm.

[Fong et al., 2009] Fong, A. T., Chaw, L. T., Keong, P. K., and Yee, P. L. (2009). Automatic web services deployment. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 07*, pages 315–319, Washington, DC, USA. IEEE Computer Society.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222.

[Fractal, 2009] Fractal (2009). The fractal project. Available at http://fractal.ow2.org/.

[Gerber et al., 2002] Gerber, A., Lawley, M., Raymond, K., Steel, J., and Wood, A. (2002). Transformation: The missing link of mda. pages 90–105. Springer.

[Gergic et al., 2002] Gergic, J., Kleindienst, J., Despotopoulos, Y., Soldatos, J., Patikis, G., Anagnostou, A., and Polymenakos, L. (2002). An approach to lightweight deployment of web services. pages 635–640.

[Globus, 2009] Globus (2009). 5.0.0 release. Specification available at http://www.globus.org/.

[Goscinski and Abramson, 2004] Goscinski, W. and Abramson, D. (2004). Distributed ant: A system to support application deployment in the grid. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 436–443, Washington, DC, USA. IEEE Computer Society.

[Gustavo et al., 2004] Gustavo, A., Fabio, C., Harumi, K., and Vijay, M. (2004). Web Services: Concepts, Architecture and Applications.

[Hall et al., 1997] Hall, R. S., Heimbigner, D., van der Hoek, A., and Wolf, A. L. (1997). The software dock: An architecture for post-development configuration management in a wide-area network. In *ICDCS*, pages 0–.

[Hall et al., 1999] Hall, R. S., Heimbigner, D., and Wolf, A. L. (1999). A cooperative approach to support software deployment using the software dock. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 174–183, New York, NY, USA. ACM.

[Hassine et al., 2002] Hassine, I., Rieu, D., Bounaas, F., and Seghrouchni, O. (2002). Symphony: un modèle conceptuel de composants métier. *Ingénierie des Systèmes d'Information*, 7(4):33–59.

[Heineman and Councill, 2001] Heineman, G. T. and Councill, W. T., editors (2001). *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Henninger, 1996] Henninger, S. (1996). Supporting the construction and evolution of component repositories. In *ICSE*, pages 279–288.

[Hnetynka, 2004] Hnetynka, P. (2004). Making deployment of distributed component-based software unified. In *AUSTRIAN COMPUTER SOCIETY*, pages 157–161.

[Hoffer et al., 2001] Hoffer, J. A., Valacich, J. S., and George, J. F. (2001). *Modern Systems Analysis and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[IBM, 2010] IBM (2010). Mtf: Model transformation framework. Available at http://www.alphaworks.ibm.com/tech/mtf.

[Isoda, 1995] Isoda, S. (1995). Experiences of a software reuse project. *J. Syst. Softw.*, 30(3):171–186.

[Jouault et al., 2008] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39.

[Jouault et al., 2006] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. (2006). Atl: a qvt-like transformation language. In *OOPSLA Companion*, pages 719–720.

[Kaur and Singh, 2009] Kaur, K. and Singh, H. (2009). Evaluating an evolving software component: case of internal design. *SIGSOFT Softw. Eng. Notes*, 34(4):1–4.

[Keeney and Cahill, 2003] Keeney, J. and Cahill, V. (2003). Chisel: A policy-driven, context-aware, dynamic adaptation framework. In *POLICY*, pages 3–14.

[Keith and Schincariol, 2006] Keith, M. and Schincariol, M. (2006). *Pro EJB 3: Java Persistence API (Pro)*. Apress, Berkely, CA, USA.

[Ketfi and Belkhatir, 2004] Ketfi, A. and Belkhatir, N. (2004). Open framework for the dynamic reconfiguration of component-based sotware. In *Software Engineering Research and Practice*, pages 948–951.

[Ketfi et al., 2002] Ketfi, M., Belkhatir, N., and Cunin, P.-Y. (2002). Adaptation dynamique concepts et expérimentations. In *Proceedings of the 15th International Conference on Software Systems Engineering and their Applications*.

[Kim et al., 2009] Kim, W., Kim, S. D., Lee, E., and Lee, S. (2009). Adoption issues for cloud computing. In *iiWAS '09: Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, pages 3–6, New York, NY, USA. ACM.

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Koziolek and Brosch, 2009] Koziolek, H. and Brosch, F. (2009). Parameter dependencies for component reliability specifications. *Electr. Notes Theor. Comput. Sci.*, 253(1):23–38.

[Lantim, 2003] Lantim, D. (2003). .*NET*. Eyrolles, Paris, France.

[Lawley and Steel, 2005] Lawley, M. and Steel, J. (2005). Practical declarative model transformation with tefkat. In *MoDELS Satellite Events*, pages 139–150.

[Lestideau et al., 2002] Lestideau, V., Belkhatir, N., yves Cunin, P., and C, A. T. B. (2002). Towards automated software component.

[Maamar et al., 2009] Maamar, Z., Sattanathan, S., Thiran, P., Benslimane, D., and Bentahar, J. (2009). An approach to engineer communities of web services - concepts, architecture, operation, and deployment. *International Journal of E-Business Research (IJEBR)*, 9(4).

[McCarthy and Dayal, 1989] McCarthy, D. and Dayal, U. (1989). The architecture of an active database management system. *SIGMOD Rec.*, 18:215–224.

[McIlroy, 1968] McIlroy, M. D. (1968). Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*.

[mediniQVT, 2010] mediniQVT (2010). medini qvt. Available at http://projects.ikv.de/qvt.

[Medvidovic and Taylor, 2000] Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93.

[Merle and Belkhatir, 2004] Merle, N. and Belkhatir, N. (2004). Une architecture conceptuelle pour le déploiement d'applications à grande échelle. In *INFORSID*, pages 461–476.

[MicoCCM, 2010] MicoCCM (2010). The mico corba component project. Available at http://www.fpx.de/MicoCCM/.

[Mikic-rakic and Medvidovic, 2002] Mikic-rakic, M. and Medvidovic, N. (2002). Architecture-level support for software component deployment in resource constrained environments. In *Proc. Component Deployment, IFIP/ACM Working Conf., LNCS 2370*, pages 31–50. Springer-Verlag.

[Miles and Hamilton, 2006] Miles, R. and Hamilton, K. (2006). *Introduction à UML2*. O'Reilly, Paris, France.

[Nau et al., 1999] Nau, D. S., Cao, Y., Lotem, A., and Muñoz-Avila, H. (1999). *SHOP: Simple Hierarchical Ordered Planner*.

[OMG, 2002] OMG (2002). Corba component model 3.0. Specification available at http://www.omg.org/spec/CCM/3.0/PDF/02-06-65.pdf.

[OMG, 2005a] OMG (2005a). *MOF QVT Final Adopted Specification*. Object Modeling Group.

[OMG, 2006a] OMG (2006a). Corba component model 4.0. Specification available at http://www.omg.org/docs/formal/06-04-01.pdf.

[OMG, 2006b] OMG (2006b). Deployment and configuration of component-based distributed application. Specification available at http://www.omg.org.

[OMG, 2005b] OMG, T. O. M. G. (2005b). Omg model driven architecture. Available at http://www.omg.org.

[OMG, 2007] OMG, T. O. M. G. (2007). Unified modeling language. Available at http://www.omg.org.

[OpenCCM, 2010] OpenCCM (2010). Openccm, the open corba component model platform. Available at http://openccm.ow2.org/.

[Pahl, 2005] Pahl, C. (2005). A conceptual architecture for semantic web services development and deployment. *Int. J. Web Grid Serv.*, 1:287–304.

[Pan et al., 2004] Pan, Y., Wang, L., Zhang, L., Xie, B., and Yang, F. (2004). Relevancy based semantic interoperation of reuse repositories. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 211–220, New York, NY, USA. ACM.

[Papamarkos et al., 2003] Papamarkos, G., Poulovassilis, A., Poulovassilis, R., and Wood, P. T. (2003). Event-condition-action rule languages for the semantic web. pages 309–327.

[Parrish et al., 2001] Parrish, A., Dixon, B., and Cordes, D. (2001). A conceptual foundation for component-based software deployment. *J. Syst. Softw.*, 57(3):193–200.

[Plasil et al., 1998] Plasil, F., Balek, D., and Janecek, R. (1998). Sofa/dcup: Architecture for component trading and dynamic updating. *Configurable Distributed Systems, International Conference on*, 0:43.

[Qureshi and Hussain, 2008] Qureshi, M. R. J. and Hussain, S. A. (2008). A reusable software component-based development process model. *Adv. Eng. Softw.*, 39(2):88–94.

[Ramakrishnan et al., 2010] Ramakrishnan, L., Jackson, K. R., Canon, S., Cholia, S., and Shalf, J. (2010). Defining future platform requirements for e-science clouds. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 101–106, New York, NY, USA. ACM.

[Rodriguez et al., 2008] Rodriguez, I. B., Drira, K., Chassot, C., and Jmaiel, M. (2008). Context-aware adaptation for group communication support applications with dynamic architecture. *CoRR*, abs/0812.3716.

[Rosenbaum and du Castel, 1995] Rosenbaum, S. and du Castel, B. (1995). Managing software reuse—an experience report. In *ICSE '95: Proceedings of the 17th*

international conference on Software engineering, pages 105–111, New York, NY, USA. ACM.

[RPM, 2010] RPM (2010). Rpm package manager. Available at http://rpm5.org/.

[Saidi et al., 2008] Saidi, R., Fredj, M., Mouline, S., Front, A., and Rieu, D. (2008). Spécification de composants métier : une approche par expression de variabilité multi-vue. In *INFORSID*.

[Samoa, 2000]Samoa (2000). Samoa : Structure d'accueil pour applications mobiles et configurables. Available at http://rangiroa.essi.fr/riveill/recherche/99-01-arc-inria-samoa.html.

[Segarra and André, 2009] Segarra, M.-T. and André, F. (2009). A distributed dynamic adaptation model for component-based applications. In *AINA*, pages 525–529.

[Sharp and Ryan, 2010] Sharp, J. H. and Ryan, S. D. (2010). A theoretical framework of component-based software development phases. *SIGMIS Database*, 41(1):56–75.

[Shaw and Garlan, 1995] Shaw, M. and Garlan, D. (1995). Formulations and formalisms in software architecture. In *Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science, Volume 1000*, pages 307–323. Springer-Verlag.

[Smacchia, 2003] Smacchia, P. (2003). *Pratique de .NET et C#*. O'Reilly, Paris, France.

[Smith and Weld, 1998] Smith, D. E. and Weld, D. S. (1998). Conformant graphplan. In *AAAI/IAAI*, pages 889–896.

[Sommer and Guidec, 2002] Sommer, N. L. and Guidec, F. (2002). A contract-based approach of resource-constrained software deployment. In *In the 1 st IFIP/ACM Working Conference on Component Deployment, volume 2370 of LNCS*, pages 15–30.

[Srivastava, 2001] Srivastava, B. (2001). Altalt: Combining graphplan and heuristic state searchthe shop planning system. *AI Magazine*, 22(3):88–90.

[Sundarraj, 2002] Sundarraj, R. P. (2002). An optimization approach to plan for reusable software components. *European Journal of Operational Research*, 142(1):128–137.

[Sutcliffe et al., 2006]Sutcliffe, A., Papamargaritis, G., and Zhao, L. (2006). Comparing requirements analysis methods for developing reusable component libraries. *J. Syst. Softw.*, 79(2):273–289.

[Szyperski et al., 2002] Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional. 2nd Edition, England.

[Troelsen, 2008a] Troelsen, A. (2008a). *Chapter 1: The Philosophy of .NET*, volume Pro VB 2008 and the .NET 3.5 Platform. APress.

[Troelsen, 2008b] Troelsen, A. (2008b). *Chapter 15: Introducing .NET Assemblies*, volume Pro VB 2008 and the .NET 3.5 Platform. APress.

[Varró and Balogh, 2007] Varró, D. and Balogh, A. (2007). The model transformation language of the viatra2 framework. *Sci. Comput. Program.*, 68(3):214–234.

[Villemur and Hammami, 2008] Villemur, T. and Hammami, E. (2008). Design and evaluation of a context-aware service deployment for collaborative sessions. *Computer Communications*, 31(17):4176–4191.

[Weide et al., 1991] Weide, B. W., Ogden, W. F., and Zweben, S. H. (1991). Reusable software components. pages 1–65.

[Weld et al., 1998] Weld, D. S., Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty & sensing actions. In *AAAI/IAAI*, pages 897–904.

[Yang et al., 2009] Yang, Q., Xu, M., and Pi, D.-C. (2009). Delegation: a language facility for dynamic software adaptation. *ACM SIGSOFT Software Engineering Notes*, 34(3):1–5.

[Yu et al., 2008] Yu, Q., Liu, X., Bouguettaya, A., and Medjahed, B. (2008). Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17:537–572.

[Zeng et al., 2004] Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30:311–327.

10
L'ANNEXE

10.1 ANNEXE 1 : ALGORITHME DE PLANIFICATION

Entrée : 1 modèle d'application **Application_model**, 1 modèle de domaine **Domain_model**, 1 modèle de stratégies **Strategy_model**

Sortie : 1 modèle de plan de déploiement **plan**

Debprog ;

Créer un modèle de plan de déploiement **plan**;

Construction d'une liste des conditions définies dans le modèle de stratégies pour lesquelles l'événement déclencheur est une installation **conditions** ;

Construction d'une liste des actions définies dans le modèle de stratégies pour lesquelles l'événement déclencheur est une installation **actions** ;

Construction d'une liste des composants définis dans le modèle d'application **composants** ;

Construction d'une liste des sites définis dans le modèle de domaine **sites** ;

Pour chaque composant appartenant à la liste **composants** **Faire**

Construction d'une sous liste de **conditions** que le composant satisfait **sousconditions** ;

Pour chaque condition de la liste **sousconditions** **Faire**

Exécution des actions_impératives reliées à la condition et construction d'une liste de sites_valides_consition pour ce composant ;

Ajout de cette liste de sites_valides_condition à la liste de tous_ les_sites_valides pour ce composant ;

*/*tous_les_sites_valides={(**site1**,site2,site3,site4,site7),(site1,site4,site7),(**site1**,site4,site5,site6,site7),(**site1**, site2, site4,site7)}*/*

Construction d'une liste de sites_valides pour ce composant vérifiant toutes les conditions ;

*/*sites_valides={**site1**,site4,site7}*/*

Construction d'une nouvelle liste de sites valides qui vérifient toutes les contraintes logicielles et matérielles du composant ;

*/*sites_valides={**site1**,site4}*/*

Exécution des actions par défaut afin d'obtenir une liste minimal de site valide ;

*/*sites-valides= {minimal = (site1); all = (site1, site4)}*/*

Faire

Pour chaque site appartenant à la liste des sites_valides **Faire**

Demander à l'utilisateur s'il souhaite effectuer un placement sur ce site, le nombre de placement permis pour un composant est connu à travers l'attribut DeployMode (Unique, Multiple) du modèle d'application ;

Ajout du placement (composant, site) au plan de déploiement plan ;

Quand l'utilisateur ajoute un **placement (composant, site)** au plan de déploiement plan, une réservation dynamique est faite sur les ressources du site ;

Faire

Farire

Retourner le plan de déploiement plan ;

Finprog ;

10.2 ANNEXE 2 : STRATEGIES DE DEPLOIEMENT COMPLETES

Deploymentstategies.xml

```
<?xml version="1.0"?>

<deploymentstrategiesmodel>
   <configuration>modelstrategies1</configuration>

   <eca-rules>

      <eca-rule>
         <ruleId>r1</ruleId>
         <typeofRule>Mandatory</typeofRule>
         ON
         <event>
            <deploymentState>INSTALL</deploymentState>
         </event>
         IF
         <condition>
            <selections>
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>EJB entity</attributeValue>
               </selection>
               OR
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>EJB session</attributeValue>
               </selection>
               OR
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>EJB message-driven</attributeValue>
               </selection>
            </selections>
         </condition>

         THEN SELECT
         <action>
            <mode>RA</mode>
            <selections>
               <selection>
                  <attributeName>
                     node.provideResources.softwareResource.softwareResourceType
                  </attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>J2EE SERVER</attributeValue>
               </selection>
         </action>
      </eca-rule>


      <eca-rule>
         <ruleId>r2</ruleId>
         <typeofRule>DEFAULT</typeofRule>
         ON
         <event>
            <deploymentState>INSTALL</deploymentState>
         </event>
         IF
         <condition>
            <selections>
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
```

```
                  <attributeValue>EJB entity</attributeValue>
               </selection>
               OR
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>EJB session</attributeValue>
               </selection>
               OR
               <selection>
                  <attributeName>component.implementation.assemblyType</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>EJB message-driven</attributeValue>
               </selection>
            </selections>
         </condition>

         THEN SELECT
         <action>
            <mode>RA</mode>
            <selections>
               <selection>
                  <attributeName>
                     node.provideResources.softwareResource.softwareResourceValue
                  </attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>JBOS4.2.3GA</attributeValue>
               </selection>
               AND
               <selection>
                  <attributeName>node.user.staff</attributeName>
                  <compareOp>=</compareOp>
                  <attributeValue>System Admin Service</attributeValue>
               </selection>

         </action>
      </eca-rule>
…
   <eca-rules>

<deploymentstrategiesmodel>
```

10.3 ANNEXE 3 : PLAN DE DEPLOIEMENT COMPLET

Solisdeploymentplan.xml

```
<DeploymentPlan>
        <name>Solisdeploymentplan</name>
        <applicaton>logichaine</application>
        <domaine>SolisDomaine</domain>
        <strategies>SolisStrategies</strategies>

        <placements>
                <placement>
                        <component> Commande_EJB-v3.jar</component>
                        <node> H2</node>
                </placement>
                <placement>
                        <component> Commande_EJB-v3.jar </component>
                        <node> H3</node>
                </placement>
                <placement>
                        <component> Commande_EJB-v3.jar</component>
                        <node> H4</node>
                </placement>
                <placement>
                        <component> Graphe-SOFA-v3.jar </component>
                        <node> Adele</node>
                </placement>
                <placement>
                        <component> Article_EJB-v3</component>
                        <node> H1</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA11</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA12</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA13</node>
```

```
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA14</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA21</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA22</node>
                </placement>
                <placement>
                        <component> Localisation_CCM-v3 </component>
                        <node> PDA23</node>
                </placement>

        </placements>
</ DeploymentPlan>
```

10.4 ANNEXE 4 : PUBLICATIONS

Conférences Internationales

Mariam Dibo and Noureddine Belkhatir, 2010. Model-driven deployment of distributed components-based software. 5th International Conference on Software and Data Technologies (*ICSOFT*). 22-24 July 2010 Athens, Greece.

Mariam Dibo, Noureddine Belkhatir, 2010. A Generic Framework for Distributed Components-based Software Systems Deployment: case study and tool description. 5 th International Conference on Enterprise Information Systems (ENASE). 22-24 July 2010 Athens, Greece.

Mariam Dibo, Noureddine Belkhatir, 2010. Defining an Unified Meta modeling Architecture for Deployment of Distributed Components-based Software applications. 12 th International Conference on Evaluation of Novel Approaches to Software Engineering (ICEIS). 8-12 June 2010 Madeira, Portugal.

Mariam Dibo, Noureddine Belkhatir, 2009. Challenges and Perspectives in the deployment of distributed components based software. 11 th International Conference on Enterprise Information Systems (ICEIS). 6-10 May 2009 Milan, Italy.

Revue/ Journal

Mariam Dibo, Noureddine Belkhatir, 2010. Méta-modèles dédiés au déploiement de logiciels à base de composants distribués. The Information - Interaction - Intelligence (I3) Journal. Décembre 2010.

Mariam Dibo, Noureddine Belkhatir, 2011. UDeploy: a Unified Deployment environment. Lecture Notes CCIS (Communications in Computer and Information Science) series .published by Springer-Verlag. ENASE Revised Selected Papers. (A paraître).

Présentations sans actes

Mariam Dibo, Noureddine Belkhatir, 2009. Méta-modèles dédiés au déploiement de logiciels à base de composants distribués. 8ème atelier sur l'Évolution, la Réutilisation et la Traçabilité des Systèmes d'Information (INFORSID__ERTSI). Toulouse, France.

Mariam Dibo, Noureddine Belkhatir, 2008. Déploiement et MDA. Séminaire ALaNOTr : Ingénierie des Modèles : nouvelles solutions et nouveaux problèmes. 4 Avril 2008, Grenoble, France.