Une démarche dirigée par les modèles pour construire les machines de déploiement des intergiciels à composants

Areski Flissi, Philippe Merle

Une démarche dirigée par les modèles pour construire les machines de déploiement des intergiciels à composants

Areski Flissi et Philippe Merle

Projet Jacquard INRIA Futurs
Laboratoire d'Informatique Fondamentale de Lille (LIFL)
UMR CNRS 8022 / Université des Sciences et Technologies de Lille (USTL)
59655 Villeneuve d'Ascq Cedex, France
Areski.Flissi@lifl.fr, Philippe.Merle@inria.fr

RÉSUMÉ. *Les intergiciels à composants permettent d'automatiser le déploiement des applications. Cette fonction, appelée machine de déploiement, instancie les applications à partir de leurs descriptions architecturales. Malheureusement, à l'heure actuelle, chaque intergiciel met en œuvre sa propre machine de déploiement. Ainsi, aucune capitalisation n'est proposée aussi bien sur le plan conceptuel qu'opérationnel. Afin de favoriser cette capitalisation, cet article propose une démarche dirigée par les modèles (à la OMG MDA) pour la construction des machines de déploiement des intergiciels à composants. Cette démarche introduit un profil UML de workflow permettant de définir des modèles de déploiement indépendants des intergiciels à composants cibles. Un tel modèle est ensuite raffiné pour chaque intergiciel cible puis le modèle obtenu est projeté, via des transformations, vers diverses plates-formes d'exécution. Les différents modèles et transformations de cette démarche sont illustrés sur la construction d'une machine de déploiement de composants CORBA mise en œuvre avec le modèle de composants Fractal.*

ABSTRACT. *Component middleware allow the automatization of applications deployment process. This function, called deployment machine, instantiates applications from their architectural descriptions. Unfortunately, currently each middleware implements its own deployment machine. Thus no capitalization is proposed as far as conceptual or implementation aspects are concerned. To promote this capitalization, this paper proposes a model driven approach to build component middleware deployment machines. This approach introduces a UML profile of workflow which allows us to define deployment models independently of any targeted component middleware. Such a model is then refined for each targeted middleware and the obtained model is mapped to different execution platforms. The models and transformations of this approach are illustrated on a CORBA Components deployment machine implemented using the Fractal component model.*

MOTS-CLÉS : *ingénierie dirigée par les modèles, déploiement, intergiciel, composant.*

KEYWORDS: *model driven software engineering, deployment, middleware, component.*

RSTI - L'objet – 11/2005. LMO'05, pages 79 à 94

1. Introduction

Aujourd'hui, pour répondre aux besoins des architectes d'applications à base de composants logiciels, les modèles de composants, tels que le modèle de composants CORBA (CCM) [OMG 02a] de l'*Object Management Group* (OMG) et le modèle Fractal [BRU 04b] du consortium ObjectWeb, sont de plus en plus évolués et tendent à se multiplier [SZY 02]. Le CCM permet l'interaction de composants répartis et hétérogènes [WAN 01], c'est-à-dire pouvant être implantés dans différents langages de programmation et supportant différents systèmes d'exploitation, protocoles réseaux et intergiciels CORBA. Fractal est quant à lui un modèle de composants extensible, léger, hiérarchique et offrant la notion de composants partagés [BRU 04a]. Les plates-formes mettant en œuvre ces modèles incluent pour la plupart une machine de déploiement dont le rôle est de permettre l'instanciation des applications en interprétant leurs descriptions architecturales. Le déploiement d'une application à base de composants logiciels consiste alors à installer les binaires des composants sur les sites d'exécution, créer les instances de composants, configurer leurs attributs métiers, lier les instances via leurs ports et les activer. À l'heure actuelle, les machines de déploiement sont réalisées de manière *adhoc* et sont spécifiques à chaque plate-forme. Il n'y a donc pas de capitalisation de concepts de haut niveau ni de réutilisation possible de ces machines pour d'autres plates-formes technologiques.

L'approche *Model Driven Architecture* (MDA) préconisée par l'OMG permet de résoudre ce problème grâce à la définition d'un modèle abstrait de l'application métier indépendant des technologies de mise en œuvre [MIL 03]. Ce modèle, appelé PIM (*Platform Independent Model*), est ensuite projeté via une transformation vers un modèle spécifique à une technologie (i.e. un PSM – *Platform Specific Model*). L'intérêt principal de l'approche MDA réside dans le fait qu'elle permet une meilleure capitalisation des applications et soustrait les concepteurs de la dépendance aux technologies d'exécution.

Ainsi, dans le but de faciliter le travail des concepteurs d'intergiciels à composants, nous proposons dans cet article une démarche dirigée par les modèles pour construire la machine de déploiement conforme à leur plate-forme et permettant de déployer les applications à base de composants logiciels tels que définis par [SZY 02]. Cette démarche introduit un profil UML de *workflow* permettant de définir des modèles abstraits (PIM) de déploiement indépendants des modèles de composants. Puis, nous raffinons un tel PIM afin d'obtenir un PIM du déploiement spécialisé pour un modèle de composants. Ensuite, une transformation s'opère sur ce modèle afin d'obtenir le modèle de déploiement spécifique à la technologie de mise en œuvre. Ce dernier est alors projeté vers une plate-forme d'exécution et nous obtenons ainsi, après génération, la machine de déploiement souhaitée.

La suite de cet article est organisée de la manière suivante. La section 2 présente la vision d'ensemble de notre démarche MDA permettant d'engendrer la machine de déploiement spécifique à un intergiciel à composants. La section 3 se concentre sur

les étapes successives et modèles de notre démarche MDA. La section 4 présente une mise en œuvre sur une plate-forme Fractal d'une machine de déploiement de composants CORBA. La section 5 discute des travaux relatifs. La dernière section conclut sur l'intérêt de notre démarche, notre contribution dans cet article et précise quelques perspectives de travail.

2. Notre démarche MDA pour construire les machines de déploiement

Notre démarche considère la conception des machines de déploiement comme étant elle-même une application « métier ». Le métier en question est le déploiement, les acteurs impliqués étant les concepteurs d'intergiciels à composants. La figure 1 présente la vision d'ensemble de notre démarche MDA aboutissant à la génération des machines de déploiement des intergiciels à composants.

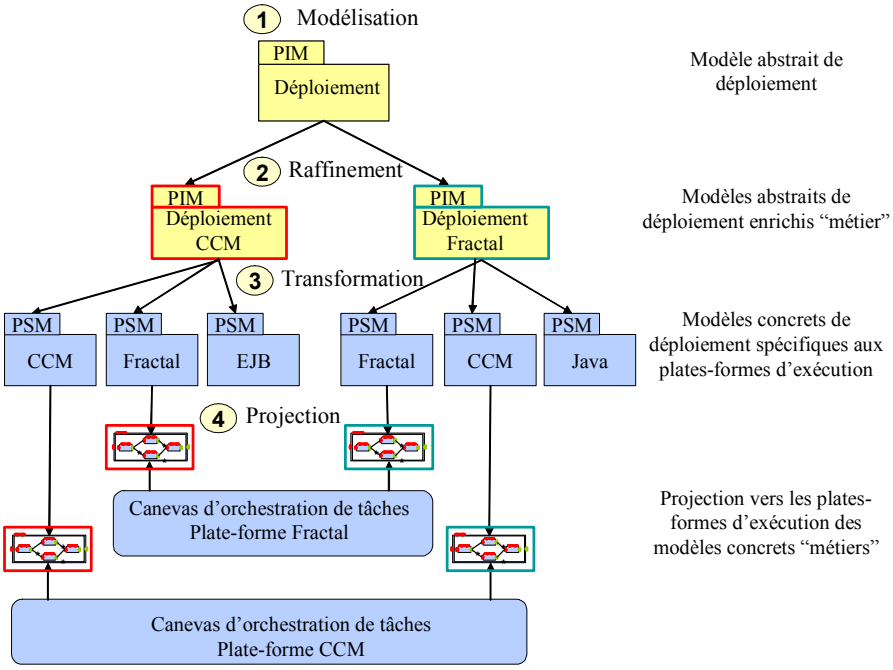


Figure 1. *Vision d'ensemble de notre démarche MDA*

La première étape (1) consiste à définir un modèle abstrait, c'est-à-dire indépendant des technologies, du déploiement des applications à base de composants. Ce modèle abstrait est donc le PIM de base de notre démarche MDA. Nous modélisons le déploiement sous la forme d'un *workflow*. En effet, nous considérons le déploiement comme un ensemble de tâches élémentaires (comme par exemple l'installation des

binaires sur les sites d'exécution ou bien l'établissement des liaisons entre les ports des composants applicatifs) à exécuter dans un ordre défini. Nous introduisons ainsi un profil UML de *workflow* nous permettant de définir des modèles de déploiement indépendants des intergiciels à composants cibles. La section 3.1 présente en détail notre proposition d'un profil UML de *workflow* illustré en section 3.2 par un exemple de modèle abstrait de déploiement.

Ensuite, l'étape suivante (2) de notre démarche consiste à raffiner ce modèle abstrait afin de l'enrichir de considérations « métiers ». Par exemple, si l'objectif final est d'engendrer une machine pour déployer des composants CORBA, alors le PIM de base doit être enrichi d'informations spécifiques au CCM. Nous parlons ici d'enrichissement métier de l'application déploiement et non de choix de la technologie de mise en œuvre du déploiement. Cette partie est détaillée en section 3.3.

Puis, conformément à l'approche MDA, l'étape suivante (3) consiste à opérer une transformation sur le PIM enrichi métier afin d'obtenir un modèle spécifique à la technologie choisie par le concepteur pour sa mise en œuvre (*i.e.* transformation PIM vers PSM). Lors de cette étape, il faut tenir compte des caractéristiques liées à la plate-forme d'exécution cible (*e.g.* CCM, Fractal, EJB, Java, etc.). Il s'agit de définir les règles de transformation des éléments du méta-modèle PIM vers les éléments du méta-modèle PSM (voir section 3.4).

La dernière étape (4) de notre démarche MDA est la projection du modèle spécifique vers la plate-forme cible. Lors de cette étape, les éléments fonctionnels de la machine de déploiement, c'est-à-dire le code métier de l'application *déploiement* pour une technologie spécifique, sont produits. Ainsi, si la plate-forme technologique cible est une plate-forme à composants, l'étape de génération consiste à appliquer un certain nombre de règles pour convertir le modèle spécifique en une configuration de composants (définition des composants, implantations, liens entre les instances, etc.). Le processus de génération de code et les règles de projection sont précisés en section 3.5.

3. Modèles et transformations de notre démarche MDA

Cette section présente en détail les différents modèles et transformations de notre démarche MDA conduisant à la génération du système exécutable « *machine de déploiement pour intergiciels à composants* ».

3.1. Le profil UML de workflow

Le déploiement d'une application à base de composants logiciels se décompose en un ensemble de tâches élémentaires. Ces tâches sont : l'installation des constituants du logiciel sur les sites d'exécution, l'instanciation des composants

applicatifs, la configuration des attributs des instances, l'établissement des liaisons entre les ports des composants, leur activation, éventuellement les actions d'enregistrement des entités (composants, fabriques, objets, etc.) dans des services d'annuaires, etc. Elles doivent s'exécuter dans un ordre donné. En effet, l'instanciation d'un composant applicatif par exemple ne peut intervenir que lorsque l'installation du binaire du composant et son chargement en mémoire ont été effectués. Nous proposons donc ici de modéliser le déploiement sous la forme d'un processus *workflow* dont les activités sont justement les tâches du déploiement alors que les flux du processus *workflow* sont des données échangées entre les tâches.

Pour ce faire, nous définissons un profil UML [OMG 03a] contenant les stéréotypes de classes *Task*, *Data* et les stéréotypes d'association *input*, *output* et *depends*. Chaque tâche abstraite de déploiement est modélisée par une classe stéréotypée «*Task*». Une classe stéréotypée «*Data*» représente un type de données particulier qu'une tâche peut requérir ou produire lors de son exécution. Chaque route du processus *workflow*, c'est-à-dire chaque relation entre deux tâches du déploiement qui partagent une donnée commune est représentée via l'utilisation d'une association entre les classes «*Task*» et «*Data*». La représentation d'une tâche qui requiert une donnée quelconque (respectivement produisant une donnée) s'effectue au moyen d'une association stéréotypée «*input*» (respectivement «*output*») entre des classes «*Task*» et «*Data*». La figure 2 illustre la représentation UML des tâches abstraites d'installation et d'instanciation, ainsi que leur relation avec la donnée abstraite *Factory* (fabrique de composant).

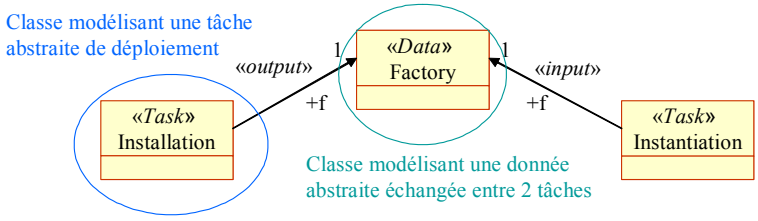


Figure 2. *Représentation UML des tâches et données de déploiement*

Enfin, les relations de dépendances en terme d'ordre d'exécution entre tâches sont exprimées via le stéréotype *depends*, comme illustré figure 3 entre les tâches qui concernent la configuration et l'activation d'un composant. L'ajout de ce stéréotype permet d'expliciter la dépendance entre deux tâches qui n'échangent pas nécessairement de données. En effet, en ce qui concerne les tâches qui partagent une donnée commune comme illustré figure 2, la dépendance est alors implicite et guidée par le processus *workflow* (une tâche qui requiert une donnée ne peut s'exécuter que lorsque la donnée concernée sera produite par la tâche précédente).

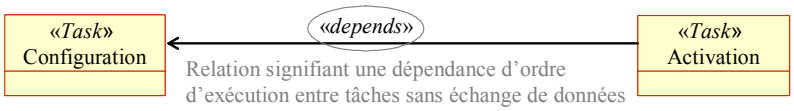


Figure 3. *Représentation UML de la notion de dépendance entre tâches*

3.2. Un modèle abstrait du déploiement

Le profil UML de *workflow* introduit précédemment nous permet de définir différents modèles abstraits du déploiement : les PIM de base pour l'application de notre démarche MDA. La figure 4 illustre un exemple de modèle PIM. Ce modèle contient les principales tâches (abstraites) élémentaires de déploiement citées dans la section précédente (classes «*Task*») ainsi que les données abstraites échangées entre ces tâches (classes «*Data*»).

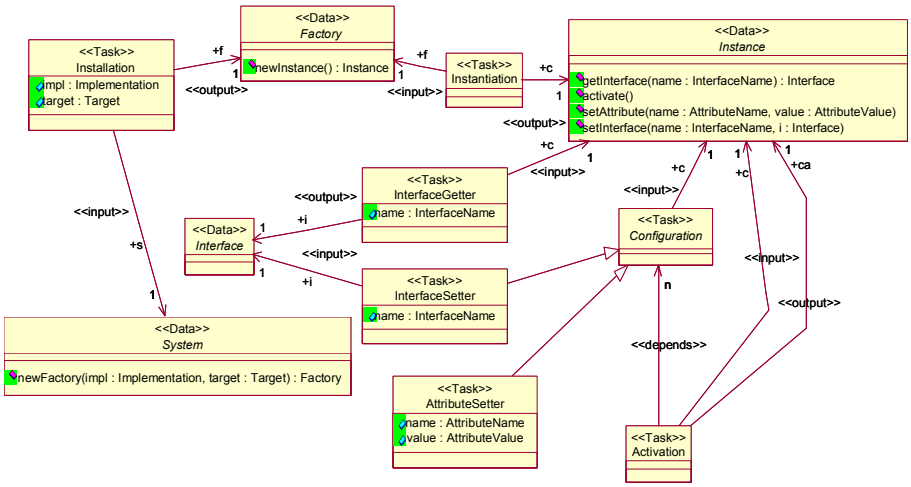


Figure 4. *Un exemple de modèle PIM du déploiement*

Les différentes classes stéréotypées «*Data*» offrent une abstraction des opérations spécifiées dans les API de déploiement des intergiciels à composants. Elles sont au nombre de quatre dans ce modèle :

– *System* : cette classe abstraite modélise une classe particulière de *bootstrap* qui fournit une méthode permettant de créer une fabrique de composant, c'est-à-dire la mise en place d'un quelconque moyen permettant de créer des instances de composants. Par exemple, cela peut signifier le téléchargement vers le site d'exécution du binaire du composant métier.

– *Factory* : cette classe abstraite modélise le concept de fabrique de composant et offre la méthode *newInstance()* permettant de créer une instance de composant.

– *Instance* : cette classe abstraite modélise la notion d'instance de composant et encapsule les opérations telles que la configuration d'un attribut métier, l'obtention d'une interface offerte, la liaison d'une interface requise avec une interface offerte par une autre instance de composant et l'activation de l'instance.

– *Interface* : cette classe abstraite modélise le concept d'interface (ou port) de composant.

Les différentes classes stéréotypées «*Task*» de ce modèle et leurs relations sont :

– *Installation* : cette classe modélise la tâche d'installation d'une fabrique de composants. Elle possède les propriétés *impl* de type *Implementation* et *target* de type *Target* représentant respectivement une implémentation (sous la forme d'une URL par exemple) et le nom du site où l'installation doit avoir lieu. Cette tâche requiert la donnée *s* de type *System* (association «*input*») et produit en sortie la donnée *f* de type *Factory* (association *«output»)*.

– *Instantiation* : cette classe modélise la tâche de création d'une instance de composant à partir d'une fabrique. Elle requiert la donnée *f* précédente (association *«input»* avec *Factory*) et produit en sortie la donnée *c* de type *Instance* (association *«output»*).

– *Configuration* : cette classe abstraite est commune à toutes les tâches de configuration des fonctionnalités (attributs, liaisons entre ports, etc.) d'une instance de composant. Une association *«input»* existe entre cette classe et la classe *Instance* modélisant le fait qu'une tâche de configuration porte sur une instance de composant.

– *AttributeSetter* : cette classe modélise la tâche de configuration d'un attribut d'une instance de composant. Elle possède les propriétés *name* et *value* représentant respectivement le nom et la valeur de l'attribut à configurer. Etant donné que la tâche de configuration d'un attribut requiert la référence de l'instance de composant à configurer, cette classe hérite de *Configuration*.

– *InterfaceGetter* : cette classe modélise la tâche d'obtention d'une interface offerte par une instance de composant. Elle possède la propriété *name* représentant le nom de l'interface offerte. Elle produit la donnée *i* de type *Interface* (association *«output»*).

– *InterfaceSetter* : cette classe modélise la tâche d'établissement de la liaison d'une interface requise par une instance de composant avec une interface fournie. Elle utilise pour cela la donnée *i* précédente (association *«input»* avec la classe *Interface*). Elle possède la propriété *name* qui représente le nom de l'interface requise et hérite également de *Configuration*.

– *Activation* : cette dernière classe modélise la tâche d'activation d'une instance de composant. Elle est en relation stéréotypée *«depends»* avec *Configuration* car l'activation dépend de l'exécution préalable des tâches de configuration comme

AttributeSetter et *InterfaceSetter*. *Activation* requiert la référence *c* de l'instance de composant à activer et produit une référence *ca* de l'instance de composant activé.

Le tableau 1 décrit de manière informelle (dans un langage proche de la syntaxe *Java*) pour chacune des tâches le comportement permettant de produire les données en sortie en fonction des données en entrée et des attributs.

Tâches	Comportement
Installation	`f = s.newFactory(impl,target)`
Instantiation	`c = f.newInstance()`
AttributeSetter	`c.setAttribute(name,value)`
InterfaceGetter	`i = c.getInterface(name)`
InterfaceSetter	`c.setInterface(name,i)`
Activation	`ca = c.activate()`

Tableau 1. *Comportement des tâches abstraites de déploiement*

Le PIM proposé ici peut être amélioré ou modifié en fonction des modèles spécifiques et plates-formes vers lesquels il sera projeté. Néanmoins, nous considérons celui-ci comme étant (à l'heure actuelle) un modèle de déploiement minimal sur lequel les concepteurs peuvent se baser pour construire les machines de déploiement des plates-formes à composants telles que CCM ou Fractal.

3.3. Le raffinement du modèle abstrait

Cette étape vise à enrichir le PIM de base du déploiement par des considérations métiers. Par « métier », nous entendons le type de machine de déploiement à construire : une machine pour déployer des composants CORBA ou des composants Fractal par exemple. Le modèle enrichi métier est un PIM car il demeure indépendant de la technologie de mise en œuvre. En effet, le modèle abstrait d'une machine de déploiement pour composants CORBA peut parfaitement être projeté vers une plate-forme technologique Fractal. En d'autres termes, la machine de déploiement d'un intergiciel à composants CORBA peut être conçue à base de composants Fractal ! Dans ce cas, elle est vue du point de vue des utilisateurs comme une boite noire dont la finalité est de déployer leurs applications CCM. Le raffinement du PIM de base doit donc prendre en compte un certain nombre d'aspects liés au modèle conceptuel des composants à déployer et est donc spécifique à ce dernier. Cependant, une démarche générale pour cette étape de raffinement du PIM existe et nous l'illustrons avec le PIM spécialisé CCM, que nous appellerons PIM pour CCM.

Les classes «*Data*» *System*, *Factory*, *Instance* et *Interface* représentant les données abstraites doivent être spécialisées (par héritage) via l'ajout de nouvelles classes «*Data*» (figure 5).

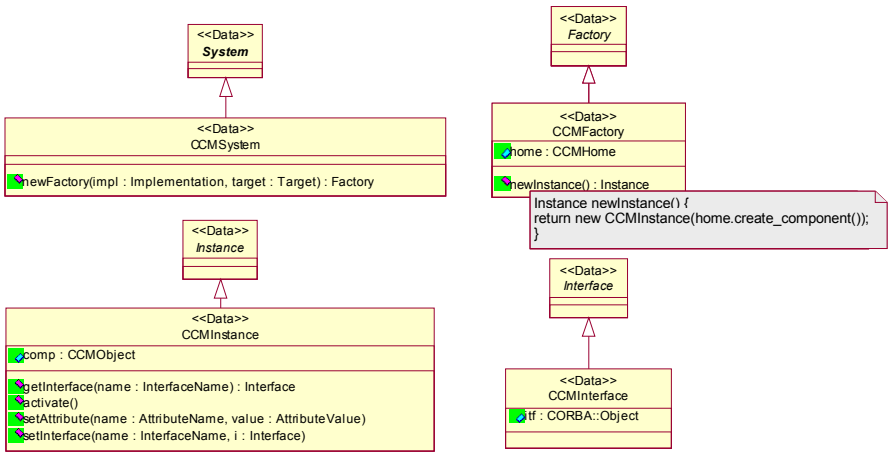


Figure 5. *Spécialisation des classes «Data» dans le PIM pour CCM*

Dans notre exemple de PIM pour CCM, *CCMFactory*, *CCMInstance* et *CCMInterface* sont ajoutés et représentent respectivement une fabrique, un composant et une interface de composant CORBA. Chacune de ces classes possèdent un attribut stockant le type spécialisé associé au modèle conceptuel de composant (en l'occurrence respectivement *CCMHome*, *CCMObject* et *CORBA::Object*). La classe *CCMSystem* spécialise quant à elle le moyen d'obtenir une fabrique de composant CORBA. Les opérations offertes par les classes représentant les données abstraites sont également spécifiées à ce stade. Par exemple, l'opération permettant de créer une nouvelle instance de composant (*newInstance()*) à partir d'une fabrique utilise l'attribut *home* de type *CCMHome* de la classe *CCMFactory* ainsi que l'opération *create_component()* offerte par l'API décrite dans la spécification CCM pour instancier un composant CORBA.

3.4. La transformation PIM vers PSM

Cette section traite de la troisième étape dans la mise en œuvre de notre démarche MDA, c'est-à-dire la transformation de notre modèle PIM enrichi vers un modèle PSM en vue de sa projection vers la plate-forme d'exécution cible. Pour illustrer cette transformation, nous prendrons comme exemple le PIM pour CCM de la section précédente pour le transformer vers un modèle spécifique pour Fractal, nommé PSM Fractal pour CCM.

Cette étape précise les règles de transformation entre les concepts du méta-modèle PIM et du méta-modèle PSM [LEM 98]. Pour cela, il nous faut dans un premier temps définir un profil UML qui sera utilisé pour décrire le PSM Fractal. Le modèle Fractal étant orienté composants, nous définissons naturellement les stéréotypes de

classes UML *FractalComponent* et *FractalInterface*, ainsi que les stéréotypes d'associations UML *FractalProvides* et *FractalUses*. Ces quatre éléments constituent un langage suffisant pour décrire un modèle spécifique à Fractal. De plus, ce choix nous permet aisément de préciser les règles de transformation entre les éléments du méta-modèle PIM et du méta-modèle PSM Fractal. En effet, le stéréotype *Task* devient *FractalComponent* représentant un composant Fractal tandis que le stéréotype *Data* devient *FractalInterface* symbolisant une interface cliente ou serveur d'un composant Fractal. En ce qui concerne le stéréotype *input* (respectivement *output*) permettant de décrire qu'une tâche *Task* requiert (respectivement produit) une donnée *Data*, il se traduit par un stéréotype *FractalUses* (respectivement *FractalProvides*) donnant le moyen d'exprimer qu'une classe *«FractalComponent»* requiert (respectivement offre) une classe *«FractalInterface»* (*e.g. D1Provider* pour une donnée de type *D1*, figure 6). De plus, les classes stéréotypées *«FractalInterface»* du modèle PSM offrent une opération permettant d'obtenir une référence vers le type de données circulant entre deux tâches (*e.g. getD1()* pour une classe *D1Provider*). Enfin, une classe particulière *TaskComponent* stéréotypée *«FractalComponent»* offrant une interface nommée *Task* stéréotypée *«FractalInterface»* est introduite lors de la transformation. Cette dernière possède une méthode *execute()* qui permet de spécialiser (du point de vue du comportement des tâches) l'implantation des classes *«FractalComponent»*. Toutes les classes *«FractalComponent»* en héritent.

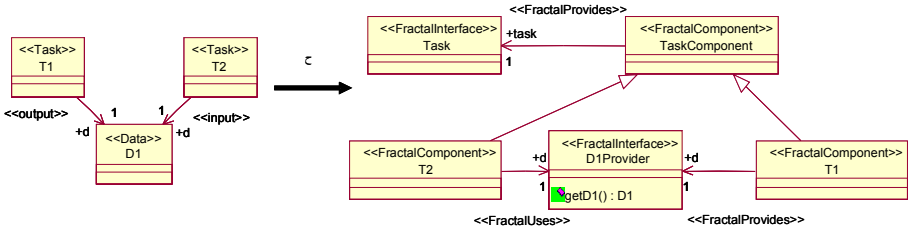


Figure 6 : *Illustration de la transformation PIM vers PSM*

Le stéréotype *depends* du profil UML de *workflow* se traduit par une utilisation conjointe de trois éléments du profil UML pour Fractal (figure 7). Une classe T2 stéréotypée «*Task*» en relation *depends* avec une classe T1 stéréotypée «*Task*» se traduit par : une classe T2 stéréotypée «*FractalComponent*» requiert (*i.e.* est en relation «*FractalUses*» avec) une classe *T1Dependency* stéréotypée «*FractalInterface*» elle-même fournie par (*i.e.* en relation «*FractalProvides*» avec) une classe T1 stéréotypée «*FractalComponent*».

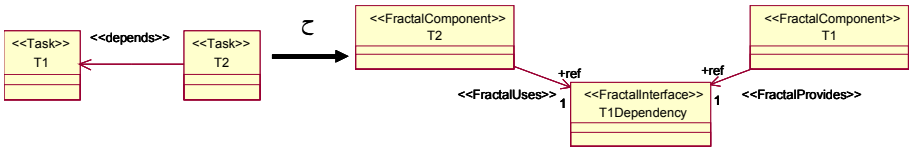


Figure 7 : *Règle de transformation PIM vers PSM du stéréotype depends*

La figure 8 illustre le PSM Fractal pour CCM obtenu à partir du PIM pour CCM proposé en section 3.3, après application des règles précédentes de transformation. Afin de ne pas alourdir la figure, seules les classes et méthodes relatives à l'étape de projection PIM vers PSM apparaissent ici (toutes les opérations des classes «*Data*» ainsi que les classes qui spécialisent ces dernières ne sont pas indiquées).

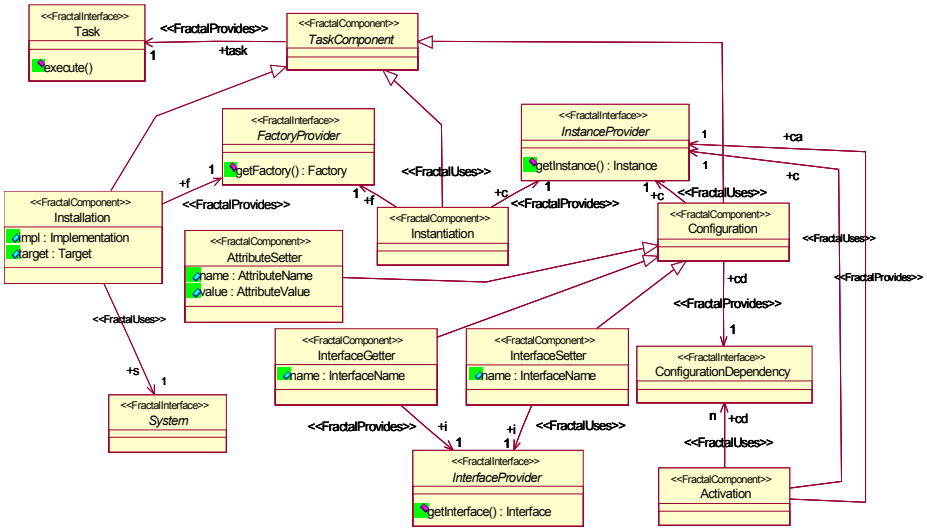


Figure 8 : *Modèle PSM Fractal pour CCM obtenu après transformation*

Par exemple, la classe *FactoryProvider* possède une méthode *getFactory()* permettant d'obtenir la référence d'un objet de type *Factory*. La relation de dépendance de la tâche *Activation* envers la tâche *Configuration* est représentée par une relation *«FractalUses»* de cardinalité *n* avec une classe *ConfigurationDependency* stéréotypée *«FractalInterface»* en relation *«FractalProvides»* avec la classe *Configuration*. De plus, la classe *TaskComponent* (en relation *«FractalProvides»* avec l'interface *Task*) est héritée par toutes les classes *«FractalComponent»* (explicitement par les classes *Installation* et *Instantiation* ou implicitement via la relation d'héritage avec *Configuration*).

Nous avons décrit ici l'étape de projection du PIM vers PSM en l'illustrant avec la technologie Fractal. Néanmoins, la démarche peut être généralisée dès lors que la plate-forme technologique cible est une plate-forme à composants. En effet, les règles de base de la transformation du modèle de tâches vers un modèle de composants peuvent s'appliquer de la même manière (*i.e.* tâche vers composant, donnée vers interface, etc.).

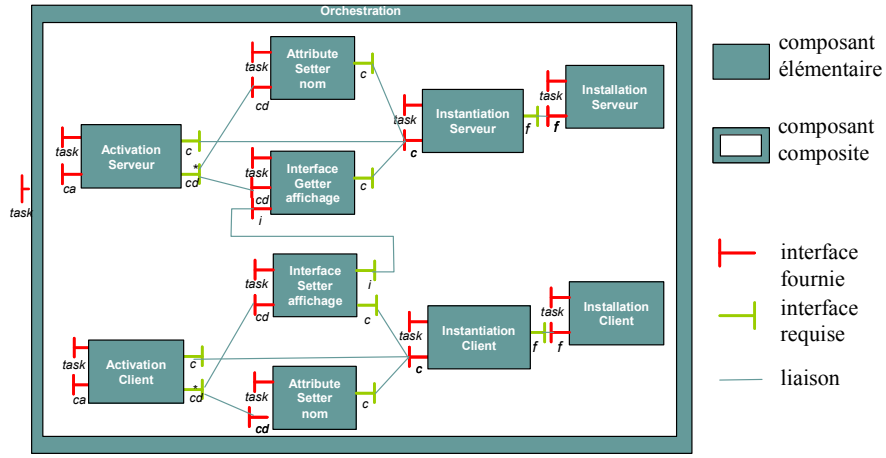
3.5. La projection du PSM vers la plate-forme technologique cible

La projection du modèle spécifique obtenu grâce à la transformation PIM vers PSM précédente est la dernière étape de l'application de notre démarche MDA. Il s'agit ici de mettre en œuvre les mécanismes aboutissant à la génération du code fonctionnel s'exécutant sur la plate-forme cible. Par exemple, reprenant le PSM Fractal pour CCM de la section précédente, il nous faut générer le code des composants Fractal dont l'assemblage constituera une machine de déploiement pour composants CORBA. Avec MDA, cette étape est facilitée par l'étape de transformation du modèle abstrait vers un modèle spécifique à une plate-forme cible car comme on peut le constater sur la figure 8 du modèle PSM Fractal pour CCM, les règles de projection des éléments du modèle vers la plate-forme cible sont intuitives. En effet, les classes stéréotypées *«FractalComponent»* sont projetées vers une définition d'un composant Fractal en Fractal ADL (*Architecture Description Language*) alors que les classes *«FractalInterface»* (et leur signature) sont projetées vers des interfaces Java. Les associations stéréotypées *«FractalUses»* et *«FractalProvides»* sont quant à elles projetées vers des déclarations en Fractal ADL permettant de caractériser les ports des composants et donc leur type. En ce qui concerne les relations d'héritage entre les classes stéréotypées *FractalComponent* (par exemple entre les éléments de la figure 8 *AttributeSetter* et *Configuration*) elles sont exprimées à l'aide de définition d'héritage entre composants dans le langage Fractal ADL. Enfin, les classes d'implantation des composants Fractal (plus précisément de l'interface *Task* des composants) implantent la méthode *execute()*. La génération du corps de ces méthodes nécessite cependant de traduire en Java les fonctions exprimées de manière abstraite dans le tableau 1.

4. La mise en œuvre sur Fractal d'une machine de déploiement CCM

Nous allons voir dans cette section un exemple de mise en œuvre concret d'une machine de déploiement. Nous reprenons pour cela l'exemple suivi tout au long des sections précédentes. L'application de notre démarche MDA nous a permis d'obtenir la personnalité métier CCM (*i.e.* le code fonctionnel des composants de déploiement) et sa mise en œuvre avec Fractal. Pour permettre son exécution, nous avons conçu un canevas d'orchestration de composants Fractal. Celui-ci utilise un composite *Orchestration* opérateur particulier de composition des composants Fractal de déploiement générés. Ce composite fournit l'interface *Task* et exécute les

tâches dans un ordre respectant leurs dépendances (voir [FLI 04] pour plus de détails). Le canevas d'orchestration de tâches permet de réaliser différentes machines de déploiement (du point de vue métier) grâce aux personnalités métiers. A terme, un canevas d'orchestration doit exister pour chaque type de plate-forme de mise en œuvre (*i.e.* pour chaque PSM), mais il est commun aux PIM métiers. La figure 9 illustre un assemblage des composants Fractal pour le déploiement d'une simple application Client/Serveur à base de composant CORBA.



f : FactoryProvider *c* : InstanceProvider *i* : InterfaceProvider *cd* : ConfigurationDependency

Figure 9 : *Un assemblage des composants Fractal de la machine de déploiement CCM*

Comme on peut le voir sur cet exemple, les tâches du déploiement sont représentées par des composants Fractal, et les relations entre les tâches comme entre *Instantiation* et *Installation* (*i.e.* une instance de composant ne peut être créée tant que l'installation de sa fabrique n'a pas été effectuée) sont représentées par les liaisons entre les interfaces des composants de déploiement. Cette figure montre la réalisation du déploiement de l'application : le canevas d'orchestration exécute les tâches d'installation des fabriques *Serveur* et *Client*, puis d'instanciation, puis de configuration des attributs et des liaisons entre les ports des composants métiers et enfin d'activation.

5. Travaux relatifs

La spécification *Software Process Engineering Metamodel* (SPEM) [OMG 02c] définit par l'OMG permet, via une extension d'UML, de modéliser les processus de développement logiciels. Notamment, considérant là encore les tâches élémentaires de déploiement comme les activités d'un processus *workflow*, le méta-modèle SPEM offre la possibilité de définir des modèles abstraits de déploiement indépendants des

intergiciels à composants. Le choix de définir notre propre langage pour décrire des modèles abstraits de déploiement a été motivé par le fait que les processus *workflow* entrants en jeu sont très simples et ne font pas intervenir de concepts complexes (comme par exemple la notion de rôle) considérés dans la spécification SPEM.

Egalement, [MER 04] propose un méta-modèle de déploiement ainsi qu'un environnement générique nommé ORYA permettant de déployer tout type d'applications logicielles, indépendamment des outils utilisés et de l'environnement cible. Notre démarche est similaire du point de vue de l'approche mais nous nous intéressons plus particulièrement au déploiement des applications conçues à base de composants. De plus, nos modèles abstraits décrivent non seulement la manière dont les constituants du logiciel seront installés, mais également comment l'application est instanciée et configurée.

Dans le cadre du projet RNTL ACCORD, un modèle abstrait de composants indépendant des technologies a été défini [BLA 04a]. Ces travaux appliquent la démarche MDA et décrivent la projection du modèle abstrait vers les plates-formes technologiques EJB et CCM. La transformation de notre PIM enrichi métier vers ce modèle abstrait de composants pourrait nous permettre d'automatiser le processus de projection de notre modèle abstrait de déploiement vers les plates-formes à composants, sans passer par l'étape de définition systématique de profils UML pour chaque modèle de composants.

Enfin, la spécification OMG « *Deployment and Configuration of Distributed Component-Based Applications* » (D&C) [OMG 03b] propose un PIM pour exprimer des assemblages de composants et le processus de déploiement indépendamment des technologies d'exécution. Ce modèle est ensuite projeté, via des transformations de modèles, vers les plates-formes technologiques cibles. Mais, à l'heure actuelle, la spécification OMG D&C est en cours de finalisation. Toutefois, notre démarche est complémentaire à D&C car notre modèle abstrait du déploiement définit une granularité plus fine en ce qui concerne les tâches du processus de déploiement des applications à base de composants.

6. Conclusion et perspectives

Notre démarche pour permettre aux concepteurs d'intergiciels à composants d'engendrer leurs machines de déploiement consiste à appliquer l'approche MDA. Pour ce faire, considérant le déploiement comme un ensemble de tâches élémentaires à exécuter dans un ordre pré-établi, nous avons introduit un profil UML de *workflow* nous permettant de définir des modèles abstraits de déploiement indépendants des intergiciels à composants visés. Un exemple de modèle abstrait a été présenté dans cet article, ainsi que les différents modèles et transformations de notre démarche MDA conduisant à la génération du système exécutable *machine de déploiement*. Nous avons également présenté une mise en oeuvre sur une plate-forme Fractal d'une machine de déploiement pour composants CORBA. Nous avons en effet conçu

un canevas d'orchestration de tâches sur lequel s'exécutent les composants de déploiement générés automatiquement. Les bénéfices de notre démarche sont nombreux et liés aux avantages de l'approche MDA. L'utilisation des modèles indépendants des plates-formes permet aux concepteurs de s'abstraire des technologies d'exécution. Leurs modèles peuvent être projetés vers différentes plates-formes d'exécution. Ils capitalisent ainsi leurs modèles abstraits de déploiement. Ceux-ci restent valables même si les plates-formes technologiques évoluent car débarrassés de toutes considérations techniques, c'est-à-dire spécifiques à une technologie.

Les perspectives pour notre travail vont dans plusieurs directions. D'une part, nous souhaitons opérationnaliser les différentes étapes de notre démarche MDA. Pour ce faire, nous comptons utiliser des outils génériques liés à MDA comme le projet ModFact [BLA 04b, ModFact]. Ces outils vont nous permettre de manipuler des référentiels MOF stockant les modèles définis dans cet article et d'automatiser les processus de transformation de modèles à l'aide d'un langage permettant d'exprimer les règles de transformation entre modèles (QVT) [OMG 02b] et d'un moteur de transformation. Une application concrète sera par exemple de générer la machine de l'infrastructure de déploiement DCI [BRI 04] de notre implantation OpenCCM [MAR 01, OPE 02] du modèle à composants CORBA. D'autre part, l'établissement de liens entre les modèles PIM métiers nous offrira la possibilité de construire une machine permettant de déployer des composants logiciels hétérogènes du point de vue des intergiciels. Cette perspective contribuera aux travaux actuels relatifs au problème de l'interopérabilité entre plates-formes à composants.

7. Bibliographie

[BLA 04a] BLANC X., CARON O., GEORGIN A., MULLER A., « Transformation de modèles : d'un modèle abstrait aux modèles EJB et CCM », *Actes de Langages et Modèles à Objets - LMO 2004*, RSTI – L'objet, vol. 10, n°2-3/2004, Lille, France, mars 2004, p. 161–174.

[BLA 04b] BLANC X., BOUZITOUNA S., GERVAIS M.P, « A Critical Analysis of MDA Standards through an Implementation: The ModFact Tool », *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, Enshede, Pays-Bas, 17-18 mars 2004.

[BRI 04] BRICLET F., CONTRERAS C., MERLE P., « OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA », *1ère Conférence sur le Déploiement et la (Re) Configuration de logiciels DECOR'04*, Grenoble, France, 28-29 octobre 2004.

[BRU 04a] BRUNETON E., COUPAYE T., LECLERCQ M., QUEMA V., STEFANI J.B., « An Open Component Model and Its Support in Java », *Proceedings of 7th International Symposium on Component-Based Software Engineering CBSE7*, Edinburgh, Ecosse, 24-25 mai 2004, vol. 3054 of LNCS, Springer-Verlag, 2004, p. 7-22.

[BRU 04b] BRUNETON E., COUPAYE T., STEFANI J.B., « The Fractal Component Model » spécification, http://fractal.objectweb.org/specification/fractal-specification.pdf, version 2.0-3, ObjectWeb Consortium, février 2004.

[FLI 04] FLISSI A., MERLE P., « Vers un environnement multi-personnalités pour la configuration et le déploiement d'applications à base de composants logiciels », *1ère Conférence sur le Déploiement et la (Re) Configuration de logiciels DECOR'04*, Grenoble, France, 28-29 octobre 2004.

[LEM 98] LEMESLE R., « Transformation rules based on meta modeling », *EDOC'98*, San Diego, novembre 1998.

[MAR 01] MARVIE R., MERLE P., GEIB J.M., VADET M., « OpenCCM : une plate-forme ouverte pour composants CORBA », *Actes de la 2ième Conférence Française sur les Systèmes d'Exploitation CFSE'2*, Paris, France, 24-26 avril 2001, p. 1-12.

[MER 04] MERLE Noëlle., « Un méta-modèle pour l'automatisation du déploiement d'applications logicielles », *1ère Conférence sur le Déploiement et la (Re) Configuration de logiciels DECOR'04*, Grenoble, France, 28-29 octobre 2004.

[MIL 03] MILLER J., MUKERJI J., « MDA Guide Version 1.0.1 », OMG TC Document omg/2003-06-01, Boston, U.S.A., juin 2003, OMG.

[ModFact] Projet ModFact, Laboratoire d'Informatique Paris 6 (LIP6). http://modfact.lip6.fr.

[OMG 02a] Object Management Group, « CORBA Components Specification, version 3.0 », OMG TC Document formal/2002-06-65, Boston, U.S.A., juin 2002, OMG.

[OMG 02b] Object Management Group, « MOF 2.0 Query / Views / Transformation RFP », OMG TC Document ad/2002-04-10, Needham, U.S.A, avril 2002, OMG.

[OMG 02c] Object Management Group, « Software Process Engineering Metamodel, v1.0 », OMG TC Document formal/2002-11-14, novembre 2002, OMG.

[OMG 03a] Object Management Group, « Unified Modeling Language, version 1.5 », OMG TC Document formal/2003-03-01, Boston, U.S.A., mars 2003, OMG.

[OMG 03b] Object Management Group, « Deployment and Configuration of Component-based Distributed Applications Specification », OMG TC Document ptc/2003-07-08, Boston, U.S.A., juillet 2003, OMG.

[OPE 02] OpenCCM – The Open CORBA Component Model Platform, 2002, ObjectWeb Consortium. http://openccm.objectweb.org.

[SZY 02] SZYPERSKI C., *Component Software: Beyond Object-Oriented Programming – 2nd edition*, New York, U.S.A., Addison Westley Longman, novembre 2002.

[WAN 01] Wang N., Schmidt D.C., O'Ryan C., « Overview of the CORBA Component Model », chapitre 31 de Component-Based Software Engineering – Putting the Pieces Together, Boston, U.S.A., Addison-Wesley, 2001, p. 557-571.