Une approche de description multi-échelles et multi points de vue pour les architectures logicielles dynamiques

Ilhem Khlif, Mohamed Hadj Kacem, Khalil Drira

Une approche de description multi-échelles et multi points de vue pour les architectures logicielles dynamiques

Ilhem Khlif [1], Mohamed Hadj Kacem[1], and khalil Drira[2,3]
[1] University of Sfax, ReDCAD, B.P.W, 3038 Sfax, Tunisia
[2] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3] Univ de Toulouse, LAAS, F-31400 Toulouse, France
ilhemkhlif@gmail.com, mohamed.hadjkacem@isimsf.rnu.tn, khalil.drira@laas.fr

Résumé

Cet article vise à présenter une approche de modélisation multi-échelles et multi points de vue pour faciliter la description et la validation des architectures logicielles dynamiques. Il s'agit de présenter une approche de conception orientée-modèle basée sur des notations visuelles permettant de décrire, par extension des notations UML, les architectures logicielles dynamiques selon différents niveaux de description. Nous décrivons un processus de raffinement progressif permettant d'automatiser le passage d'un modèle générique décrivant un point de vue donné d'une échelle à un modèle spécifique décrivant ce point de vue sous une autre échelle. Afin de garantir la compatibilité entre les différents niveaux de description, nous adoptons une technique de transformation orientée règles. Les règles gèrent le raffinement d'une échelle à une autre et également le passage d'un point de vue à un autre dans une même échelle. La modélisation des architectures logicielles dynamiques génère des modèles qui doivent être valides par rapport à leurs méta-modèles. L'approche que nous proposons est basée sur des règles de validation entre les différentes échelles et les différents points de vue de la même échelle. Cette validation assure la validité du modèle architectural. Nous présentons également l'interface d'un plug-in Eclipse que nous avons développé pour implémenter et expérimenter notre approche. Nous illustrons notre approche par son application à une étude de cas modélisant les systèmes de support des opérations d'intervention d'urgence (OIU).

Mots Clés:

Architectures logicielles dynamiques, descriptions multi-échelles, descriptions multi points de vue, raffinement d'architecture, notations UML.

Abstract

This paper aims to presenting an approach of modeling multi-scale and multi point of view to facilitate the description and the validation of dynamic software architectures. We present a model-based design approach founded on visual notations allowing to describe, by extending UML notations, dynamic software architectures for different description levels. We describe a progressive process of refinement to automate the transition from a generic model describing a given point of view at a given scale to a specific model describing this point of view at another scale. To ensure compatibility between the different levels of description, we adopt a rule-oriented transformation technique. The rules manage the refinement of a given scale model to another and also the transition from a point of view to another at the same scale. The modeling of dynamic software architectures generates models that must be valid with respect to their meta-models. The approach that we propose is based on validation rules between different scales and different points of view at the same scale. This validation ensures the validity of the architectural model. We also present the interface of an Eclipse plug-in that we developed to implement and experiment our approach. We illustrate our approach by its application to a case study modeling Emergency Response and Crisis Management Systems(ERCMS).

Keywords

Dynamic Software Architectures, multi-scales descriptions, multi points of view descriptions, architecture refinement, UML notations.

1. INTRODUCTION

L'architecture logicielle a émergé comme un axe de recherche important de l'ingénierie logicielle. Elle fournit la description de haut niveau de la structure d'un système logiciel de façon de réduire sa complexité.
L'architecture logicielle inclut la description des éléments à partir desquels les systèmes sont construits, les interactions entre ces éléments, les patrons qui guident leur composition et les contraintes sur ses patrons (Garlan et Shaw, 1994)[10]. En d'autres termes, l'architecture logicielle décrit d'une manière symbolique et schématique les différents éléments d'un ou de plusieurs systèmes informatiques et leurs interactions.

Ainsi, la conception architecturale se base sur des patrons récurrents d'organisation appelés styles architecturaux. La modélisation de l'architecture logicielle génère des modèles en se basant sur le profil UML (Unified Modeling Language). En effet, la plupart des systèmes logiciels doivent subir de différentes modifications durant leurs cycles de vie afin de faire face aux nouveaux besoins des utilisateurs. L'enjeu est donc de modéliser ces modifications architecturales tout en préservant la validité de l'architecture par rapport à son style.

UML est un langage de modélisation qui décrit un système en utilisant une notation graphique orientée objet. Il a une syntaxe semi-formelle et une sémantique.
Pour modéliser un système, UML fournit plusieurs modèles exprimés sous forme de diagrammes (de classes, d'états, de packages, de cas d'utilisation, de composants et de déploiement). Une description architecturale doit être comprise et manipulée par plusieurs intervenants qui ont peu de connaissance dans le domaine des spécifications formelles. C'est l'une des raisons pour lesquelles plusieurs tentatives ont été faites pour utiliser ou étudier la possibilité d'utiliser UML comme un langage de description des architectures. L'autre raison est qu'UML est un standard de l'OMG qui couvre, par ses outils de support, toutes les phases d'un cycle de développement (Muller et Gaertner, 2004)[18] et (Piechocki, 2007)[21].

Un aspect clé de la conception de tout système logiciel est le concept de raffinement de l'architecture.
Le raffinement d'une bonne architecture garantit l'intégrité et la cohérence de l'architecture logicielle, permet de réduire les coûts et d'améliorer la qualité du logiciel.
Cependant, le raffinement de l'architecture logicielle est considéré comme une étape en cours de la recherche à cause de la complexité de la conversion de l'architecture abstraite à l'architecture spécifique. Le raffinement de l'architecture est devenu donc un domaine de recherche important en génie logiciel pour la description de l'architecture logicielle.

Nous nous intéressons particulièrement dans nos travaux de recherche à la modélisation des architectures logicielles dynamiques. Cet article vise à présenter une approche de modélisation multi-échelles et multi points de vue pour faciliter la description et la validation des architectures logicielles dynamiques. Il s'agit de présenter une approche de conception orientée-modèle basée sur des notations visuelles permettant de décrire, par extension des notations UML, les architectures logicielles dynamiques selon différents niveaux de description.

Nous nous basons dans notre recherche sur le langage UML puisqu'il est le langage de modélisation le plus répondu et qui s'adapte grâce à ses extensions à la modélisation des architectures dynamiques.

La principale contribution de notre travail est de décrire un processus de raffinement progressif permettant d'automatiser le passage d'un modèle générique décrivant un point de vue donné d'une échelle à un modèle spécifique décrivant ce point de vue sous une autre échelle. L'objectif de notre travail est de générer un système qui peut supporter différents niveaux de description. Un processus de raffinement progressif est nécessaire pour partir d'un modèle générique d'un niveau d'abstraction N-1 à un modèle plus spécifique d'un niveau N. Afin de garantir la compatibilité entre les différents niveaux de description, nous adoptons une technique de transformation orientée règles. Les règles gèrent le raffinement d'une échelle à une autre et également le passage d'un point de vue à un autre dans une même échelle. En effet, une échelle peut être définie comme un modèle qui donne des détails supplémentaires de la conception et décrit également un niveau ou une couche dans un système de communication. De même, une échelle peut avoir plusieurs points de vue procédés par un raffinement du modèle pour exprimer plus de détails.

La modélisation des architectures logicielles dynamiques génère des modèles qui doivent être valides par rapport à leurs méta-modèles. L'approche que nous proposons est basée sur des règles de validation entre les différentes échelles et les différents points de vue de la même échelle. Cette validation assure la validité du modèle architectural.

Nous présentons également l'interface d'un plug-in Eclipse que nous avons développé pour implémenter et expérimenter notre approche. Nous nous intéressons dans nos domaines d'applications à la modélisation des systèmes collaboratifs dans des environnements communicants (SCC) et nous illustrons notre approche par son application à une étude de cas modélisant les systèmes de support des opérations d'intervention d'urgence (OIU) et s'inscrivant dans le projet ROSACE (RObots et Systèmes Auto-adaptatifs Communiquants Embarqués) (Drira, 2009)[15].

La suite de l'article est organisée de la façon suivante. La section 2 est dédiée pour l'état de l'art présentant une étude synthétique de travaux similaires à notre approche. La démarche générale de l'approche est décrite puis détaillée à travers une étude de cas dans la section 3. La section 4 propose l'implémentation de notre travail sous l'environnement Eclipse. Finalement, la section 5 conclut et décrit les perspectives de notre travail.

2. ÉTAT DE L'ART

Il existe deux domaines de recherche relatifs à notre travail: les architectures logicielles dynamiques (figure 1) et les méthodes de raffinement de l'architecture (figure 3).

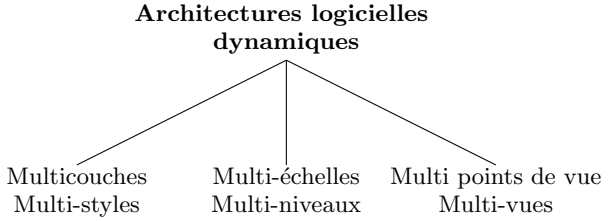

Figure 1: Description des architectures logicielles dynamiques

Dans le cadre UML, la modélisation des architectures logi-

cielles multi-niveaux a donné lieu à un certain nombre d'études (figure2).

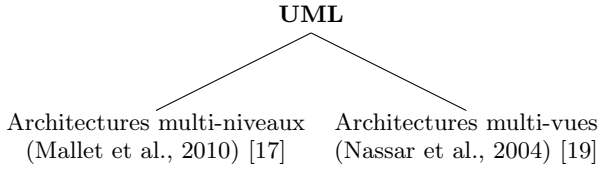

Figure 2: Description des architectures logicielles avec UML

Les travaux de (Mallet et al., 2010)[17] ont utilisé le paradigme multi-niveau pour réduire la complexité des systèmes logiciels à grande échelle. Ils ont proposé une approche par typage fort permettant de transformer automatiquement le modèle métier en un profil UML.
Cette approche est détaillée à travers deux études de cas. Le premier est un extrait du modèle de temps de Marte (Modeling and Analysis of Real-Time and Embedded Systems) de l'OMG (Object Management Group). Le deuxième permet d'exprimer en UML des résultats de mesures physiques. Ces travaux sont évalués par cette approche automatique de modélisation multi-niveau permettant d'introduire quatre niveaux d'instanciation et d'appliquer des métriques de façon systématique sur de gros projets (comme Marte). Néanmoins, ils n'ont pas proposé les métriques nécessaires pour quantifier objectivement la complexité des systèmes multi-niveaux. Les travaux de (Mallet et al., 2010)[17] ont modélisé les architectures multi-niveaux à l'aide d'un nouveau profil UML appelé DomainSpecification. Mais, ils n'ont pas implémenté ce profil avec un outil spécifique. Le concept de raffinement entre les niveaux d'une architecture est totalement absent dans ce travail.

De même, l'aspect multi-vu des architectures logicielles dynamiques a été étudié par les travaux de (Nassar et al., 2004) [19]. Ils ont développé une méthodologie de conception dont le noyau est un profil UML, appelé VUML (View based Unified Modeling Language), qui supporte la construction de composants de conception multi-vues. Ce travail a pour objet de spécialiser UML afin qu'il supporte la notion de vue/-point de vue de l'analyse jusqu'au déploiement d'une application. Cette spécialisation se traduit par l'introduction d'un ensemble de stéréotypes regroupé sous forme de profil UML.
Le méta-modèle de VUML étend celui d'UML avec de nouveaux stéréotypes : MultiViewsClass, AbstractView, Base, View, ViewExtension et ViewDependency.
Ainsi, les travaux de (Nassar et al., 2004)[19] ont présenté le patron d'implémentation pour la génération de code multi-cibles et ont adapté l'outil Objecteering/UML Modeler (qui fait partie de l'atelier Objecteering/UML) pour tenir compte des contraintes de modélisation propres à des projets.

Ce travail a mis en oeuvre l'outil support de VUML à travers deux profils Objecteering qui sont le profil Objecteering VUML qui permet de mener une modélisation selon VUML et de vérifier la conformité des diagrammes avec la sémantique de VUML et le profil Objecteering qui permet la génération du code Java à partir d'une modélisation VUML. Mais, il n'a pas précisé un processus de raffinement entre les différents points de vues dans ses profils générés.

Diverses approches visant à trouver des raffinements dans les architectures multi-niveaux ont été présentées dans la littérature. Dans un raffinement progressif, une séquence d'étapes à partir d'une spécification abstraite de l'architecture conduit à une implémentation concrète centrée sur le modèle architectural. Ces étapes de raffinement peuvent être effectuées selon deux directions: horizontale et verticale. Le raffinement horizontal permet d'introduire de nouvelles vues plus détaillées. Chaque ajout de détails à un modèle entraîne un nouveau modèle qui raffine le précédent. Quant au raffinement vertical, il permet de passer d'un niveau abstrait vers un niveau plus concret. Chaque étape de raffinement conduit généralement à une description plus détaillée de l'architecture (Oquendo, 2006) [20]. Pour un processus de raffinement de l'architecture logicielle, il existe fondamentalement deux méthodes possibles, la méthode semi-formelle et la méthode formelle (figure3).

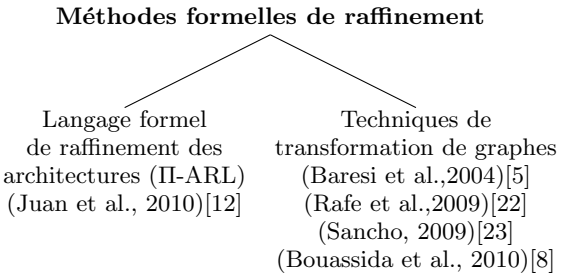

Figure 3: Méthodes de raffinement de l'architecture

Parmi les travaux relatifs au raffinement des architectures en se basant sur la méthode formelle, nous citons les travaux de (Juan et al., 2010)[12] qui vise à répondre à l'exhaustivité et la cohérence de l'application dans un processus de raffinement des architectures logicielles. L'approche proposée tente de se concentrer sur le composant comme un élément de base pendant le processus de raffinement de l'architecture et de décrire formellement les architectures logicielles avec un langage de raffinement formel Π-ARL.

Ainsi, les travaux de (Juan et al., 2010)[12] ont proposé un framework pour une approche qui se concentre sur le raffinement des composants. La description de l'architecture avec Π-ARL interprète correctement les exigences de performance des composants. Dans chaque cycle de raffinement, l'architecte décompose la couche précédente des composants en fonction de l'analyse des fonctions des composants, puis, il obtient un nombre de composants et d'interactions entre eux. Ce cycle poursuit jusqu'à ce que les composants en béton soient indécomposables.

Cette approche a assuré la cohérence sémantique dans le processus de raffinement d'architecture entre les couches et elle a montré des détails sur le niveau d'implémentation de l'architecture pour combler le fossé entre l'architecture et l'exécution. Mais, cette méthode formelle n'a pas fourni une technologie conçue avec ce langage de raffinement pour soutenir le développement des systèmes logiciels complexes.

D'autres travaux relatifs au raffinement des architectures ont abordé aussi les techniques de transformation de graphes parmi lesquels nous citons les travaux de (Baresi et al.,2004)[5] et (Rafe et al.,2009)[22].

Les travaux de (Baresi et al.,2004)[5] ont adressé le problème du développement des architectures dynamiques comme une tâche complexe généralement conduite par une approche par étapes de raffinement.

Dans le domaine de recherche de l'architecture logicielle, les styles architecturaux sont utilisés pour décrire les familles d'architectures par types de ressources communes, les schémas de configuration et des contraintes.
Ce travail définit les relations de raffinement entre les styles abstraits et concrets permettant de vérifier le raffinement correct des deux architectures données. Pour ce là, il introduit une technique formelle afin de vérifier et de construire des raffinements des architectures dynamiques. Cette technique utilise des systèmes de transformation de graphe pour modéliser les styles architecturaux pour les différents niveaux d'abstraction et de la plateforme représentée par des scénarios de reconfiguration comme des séquences de transformation de graphe. Le style basé sur l'abstraction et les mappages de raffinement ont été introduits pour raffiner les scénarios de reconfiguration automatique, tout en préservant l'exactitude sémantique. L'architecte de style définit également une correspondance entre le graphe de type et des parties du méta-modèle UML qui peut être utilisé pour convertir des diagrammes UML dans les représentations graphiques.

Les architectes d'applications peuvent utiliser le modèle de leurs architectures conventionnelles des diagrammes UML, les valider et les raffiner à l'aide de cette approche qui définit les règles UML et le mappage. Mais, cette approche nécessite deux types d'intervention humaine. Les gens qui sont compétents dans la transformation de graphe à la fois et les styles architecturaux pour concevoir des systèmes de transformation de graphe avec les types de graphes, les contraintes et les règles spécifiques.

D'autres travaux, comme (Rafe et al., 2009)[22], ont proposé une approche automatisée permettant de raffiner les différents modèles dans une plate-forme middleware spécifique dans le but de diminuer le temps de commercialisation, tout en augmentant la qualité des logiciels.
En effet, concevoir une approche de raffinement du modèle sans tenir compte de l'automatisation est une tâche complexe devant faire face aux exigences fonctionnelles et non fonctionnelles: la modélisation et l'application des exigences fonctionnelles qui doivent se conformer à la plate-forme middleware.

Cette approche formelle et automatique de raffinement des plateformes indépendamment des modèles permet d'obtenir des plateformes plus spécifiques. Les systèmes de transformation de graphe se sont révélés être une solution pour la transformation de modèle et de raffinement. Cette proposition suggère une solution par étapes au raffinement du modèle. Pour chaque niveau d'abstraction, un style doit être conçu comme un schéma graphique et des règles de graphe.

Ces deux dernières approches ont conçu des modèles multi-niveaux et multi-styles pour les systèmes logiciels en se basant sur leurs propres méthodes de raffinement. Mais, elles n'ont pas détaillé la modélisation de leurs approches à travers une étude de cas et leurs implémentations avec un outil de développement pour concrétiser ces travaux.

Dans le même axe de recherche sur les architectures multi-niveaux à base des techniques de transformation de graphes, les travaux de (Sancho, 2009)[23], (Bouassida et al., 2010)[8]et (Bouassida, 2011)[6] ont donné naissance à une approche de modélisation multi-niveau pour les systèmes collaboratifs ubiquitaires.

L'objet de ces travaux est de spécifier les problèmes de ces systèmes ainsi que de leur proposer un cadre conceptuel. Afin de modéliser des architectures logicielles complexes multi-niveaux, ils ont présenté deux procédures permettant de transformer des modèles entre les niveaux: le raffinement et la sélection. Le raffinement permet de calculer l'ensemble de modèles d'un niveau considéré qui implantent le modèle donné de niveau supérieur. La sélection permet de choisir le bon modèle parmi tous les modèles possibles d'un niveau considéré représentant l'architecture qui sera effectivement déployée. Ces travaux se sont basés sur les techniques de transformation de modèles basés sur les ontologies notamment les techniques de transformation de graphes pour le raffinement des configurations architecturales dans un système multi-niveau (niveau application, niveau collaboration et niveau middelware).

Ces travaux ont présenté un exemple d'application de l'approche qui est un jeu collaboratif développé dans le cadre du projet européen UseNet. Il est présenté à travers un modèle de niveau application et des règles permettant son raffinement au niveau collaboration. Ces travaux ont défini un cadre algorithmique générique de reconfiguration architecturale multi-niveaux qui a été appliquée au cas de la communication et de la coopération de groupe. puis implanté selon les techniques formelles. Ces techniques nécessitent forcément une formation particulière et des aptitudes mathématiques.

D'autres travaux de recherche ont présenté des méthodes de raffinement sur les architectures multi-styles. Les travaux de (Georg Jung et John Hatcliff, 2009)[13] ont défini le raffinement des styles en utilisant l'héritage multiple. Il s'agit de définir les relations entre les styles pour connecter leurs interfaces et préciser la superposition ou l'abstraction de contraintes. (Georg Jung et John Hatcliff, 2009)[13] ont décrit des opérations sur les méta-modèles fournies par CALM, un framework de méta-modélisation des systèmes multi-niveaux du langage d'architecture CADENA basé sur Eclipse, qui permettent des descriptions de raffinements de plates-formes architecturales. Ils ont illustré les principes de CALM en utilisant un système formulé en termes d'un modèle de composants hybride qui intègre trois cycles architecturaux y compris un style pour nesC, un modèle de composants et d'infrastructures connexes qui a été utilisé pour la construction de réseaux de capteurs sans fil, le style de planification de haut niveau et le style de la couche physique nécessaire dans la construction de système de systèmes.

L'architecture est en couches et les développeurs doivent se conformer à ces contraintes de superposition (par exemple, l'implémentation de chaque composant de la couche de planification doit être exprimée en nesC et chaque composant nesC associé à la catégorie du Hardware doivent être décrites en utilisant le style de la couche physique afin d'éviter l'architecture qui se dégrade au cours du temps). L'encapsulation est utilisée comme un mécanisme d'abstraction pour les deux (composants et connecteurs) qui peut impliquer un changement dans les styles d'architecture pour représenter une frontière d'abstraction. Ainsi, ces travaux de recherche ont manipulé le style CALM permettant aux architectes de créer des environnements de modélisation qui contrôlent l'évolution ou le raffinement d'une architecture d'une plate-forme indépendante d'usage général de description à un style qui contient plus de détails sur la plate-forme sous-jacente.

L'ingénierie logicielle n'a cessé de produire de nouvelles approches et de nouveaux paradigmes qui présentent l'intérêt

de permettre le raisonnement sur des systèmes logiciels complexes à large échelle (figure4).

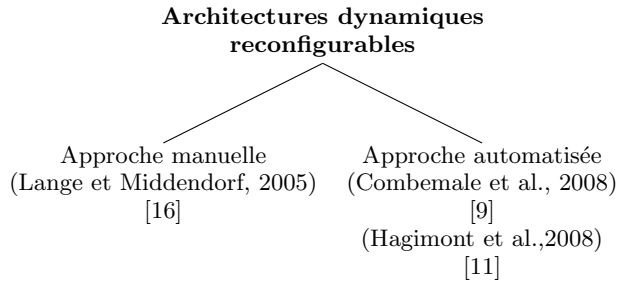

Figure 4: Architectures dynamiques reconfigurables

Une approche de conception des architectures reconfigurables à deux niveaux a été proposée par (Lange et Middendorf, 2005)[16]. Trouver la granularité optimale est un problème de conception central pour les architectures reconfigurables à deux niveaux. La solution proposée est une heuristique basée sur la fusion des deux ensembles d'une partition qui a des effets sur les coûts de l'hyper-reconfiguration. De même, deux approches automatisées de (Combemale et al., 2008)[9] et (Hagimont et al.,2008)[11] ont étudié les architectures logicielles à base de composants. Elles ont étudié la conception et l'implémentation d'un système autonome appelé Tune qui s'appuie sur un modèle de composants fournissant un soutien pour l'encapsulation de logiciels décrivant l'architecture du logiciel à gérer et à son déploiement dans l'environnement physique. Les expériences de (Combemale et al., 2008)[9] avec Tune ont mis l'accent sur l'utilisation de XML (Extensible Markup Language) et UML pour profiter de nombreux outils open source par exemple TOPCASED basé sur Eclipse Toolkit pour la description des architectures. Ces expériences ont confirmé l'intérêt d'élever le niveau d'abstraction et de spécialiser la sémantique UML selon les exigences du champ considéré. (Hagimont et al., 2008)[11] ont développé un outil pour la description de wrapper basé sur le langage de description d'encapsulation (Wrapping Description Language) pour spécifier le comportement des encapsulations. Ces travaux ont étudié l'aspect dynamique pour les architectures logicielles à base de composants. Seulement, ils n'ont pas introduit un processus clair pour la modélisation multi-niveau et le raffinement de ces architectures reconfigurables.

Selon cette étude, plusieurs travaux ont été menés pour la modélisation multi-niveau des architectures logicielles par la définition d'une notation visuelle et unifiée en se basant sur le profil UML. Mais, ces travaux qui ont intérêt essentiellement à la description du paradigme multi-niveau n'ont pas précisé la relation entre les différents niveaux de l'architecture dans un système logiciel multi-niveau. Le concept de raffinement a été absent pour les méthodes semi-formelles. Les techniques formelles et plus précisément les travaux menés sur les transformations de graphe permettent de raffiner l'architecture logicielle mais ces travaux exigent toujours un certain niveau d'aptitude mathématique. Nous avons noté également que peu de travaux présentent un processus clair, un environnement adéquat pour la description multi-niveau des architectures logicielles dynamiques.

3. DESCRIPTION DES ARCHITECTURES MULTI-ÉCHELLES ET MULTI POINTS DE VUE

3.1 Application de l'approche

Après cette analyse préalable des différents travaux existants, nous présentons notre approche de description multi-échelles et multi points de vue pour les architectures logicielles dynamiques qui cherche à combiner plusieurs formalismes en profitant de leurs avantages et en essayant d'apporter des solutions à leurs limites. Nous allons considérer un nombre d'échelles fixe et préciser les notations pour chaque échelle considérée. Nous proposons la description de notre méthodologie à travers un tableau à deux dimensions dont les lignes présentent les échelles et les colonnes présentent les points de vue d'une échelle considérée (voir figure 5).

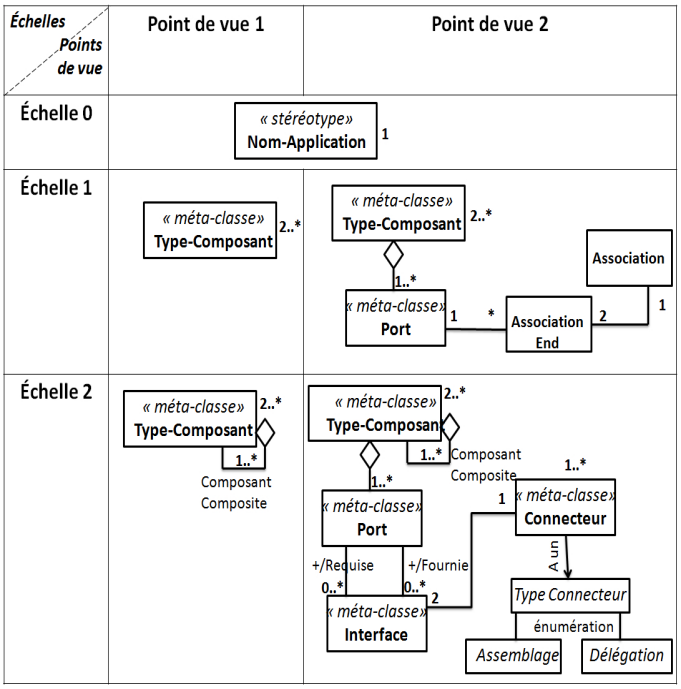

Figure 5: Application de l'approche

Nous considérons à l'échelle 0 toute l'application et nous la représentons sous forme d'un seul composant en lui attribuant un nom de l'application. Nous procédons à un raffinement du modèle pour passer à une nouvelle échelle qui permet de fournir des détails supplémentaires sur le composant générique de l'échelle 0. L'échelle 1 admet deux points de vue : Le premier point de vue représente les types de composants inclus dans l'application entière. Nous utilisons le type de composant comme étant la définition abstraite d'une entité logicielle pour la description de notre style architectural. Le deuxième point de vue procédé par un raffinement du modèle de la même échelle offre plus de détails sur les relations entre ces types de composants. Nous présentons ces relations selon le deuxième point de vue de l'échelle 1 sous forme d'associations entre les types de composants. Finalement, nous présentons à l'échelle 2 deux points de vue. Un premier raffinement donne lieu au point de vue 1 de l'échelle 2 qui représente un composant composite

pour chaque type de composants de l'application. Un deuxième raffinement met en relief un point de vue 2 de l'échelle 2 et permet de passer aux détails de chaque composant composite des types de composants en ajoutant les connexions établies entre eux. En effet, chaque association reliant deux types de composant au point de vue 2 de l'échelle 1 est raffinée en une connexion plus détaillée au point de vue 2 de l'échelle 2. Le deuxième point de vue consiste donc à identifier pour chaque composant composite, la liste de ses ports et de ses interfaces ainsi que la liste des connexions. Pour chaque connexion, il faut identifier le type de composant source et le type de composant destinataire. De ce fait, nous pouvons extraire les règles de passage entre les échelles et les points de vue de la même échelle.

- Raffinement du modèle entre les différentes échelles
  L'échelle 0 considère l'application entière représentée par le stéréotype unique ≪Nom-Application≫ qui décrit le nom de l'application à spécifier. L'unicité est représentée par la cardinalité [1]. Le passage d'une échelle à une autre permet de fournir des détails supplémentaires sur le composant générique ≪Nom-Application≫. L'échelle 1 comprend la méta-classe `Type-Composant`. Elle spécifie l'ensemble des types de composants qui constituent l'application. Cette spécification est décrite en utilisant la notation UML 2.0. La multiplicité est représentée par la cardinalité [2..*] car une application comprend au moins deux types de composants. L'échelle 2 comprend la méta-classe `Type-Composant` éventuellement composée de plusieurs composants composites décrits par la méta-classe `Composant-Composite`. Ces derniers représentent une partie modulaire des types de composants de l'application et sont restreints à la cardinalité [1..*]. Le passage d'une échelle à une autre permet de fournir plus de détails sur les propriétés de types de composants.

- Raffinement du modèle de la même échelle (différents points de vue)
  Le passage d'un point de vue 1 à un point de vue 2 de la même échelle signifie l'ajout des relations établies entre les types de composants à l'échelle 1 et entre les composants composites d'un type de composant à l'échelle 2.

  A l'échelle 1, chaque `Type-Composant` interagit avec son environnement par l'intermédiaire de ses ports. La cardinalité [1..*] entre les méta-classes `Type-Composant` et `Port` exprime qu'un composant peut avoir un ou plusieurs ports. Une première règle gère le passage du point de vue composant au point de vue association.

  A l'échelle 2, le premier point de vue consiste à identifier les composants composites des types de composants. Le deuxième point de vue consiste à spécifier davantage les détails sur le type de chacune des associations établies entre les types de composants. Une deuxième règle gère le passage de point de vue composant au point de vue connexion qui sera plus spécifique que le point de vue association. Chaque `Type-Composant` a une ou plusieurs interfaces fournies et/ou requises exposées par l'intermédiaire de ports.

  La méta-classe `Port` représente le point d'interaction pour un composant. La méta-classe `Interface` représente l'interface d'un composant. Une interface peut être soit de type fourni '+=Fournie' ou requis '+=requise'. La méta-classe `Connecteur` définit un lien qui rend possible la communication entre deux ou plusieurs composants.

  La méta-classe `Type Connecteur` spécifie deux types de connecteurs : les connecteurs de délégation et les connecteurs d'assemblage.
  Un connecteur de type délégation qui exprime un lien entre deux composants partant d'une interface requise vers une interface requise ou d'une interface fournie vers une interface fournie.
  Un connecteur de type assemblage qui exprime un lien entre deux composants partant d'une interface requise vers une interface fournie (HajdKacem et al., 2010) [14]. Pour chaque composant composite, il faut également identifier la liste de ses ports et de ses interfaces ainsi que la liste des connexions établies entre eux.

3.2 Illustration de l'approche : Étude de cas des OIU

Afin d'illustrer notre approche, nous considérons les systèmes de gestion d'intervention d'urgence et des crises (ER-CMS) que nous traitons en détail à travers l'étude de cas des opérations d'intervention d'urgence (OIU).

Les OIU impliquent des groupes structurés de participants communicants et coopérants afin de réaliser une mission commune (Par exemple: lutter contre les incendies et sauver les vies humaines). Les éléments de l'architecture possèdent différents rôles et disposent de ressources inégales en énergie et en capacités de communication. Ils sont déployés sur des machines fixes et mobiles et communiquent à travers des réseaux filaires et sans fil (Drira, 2010)[7].

Les relations de communication entre les participants évoluent tout au long de la mission pour s'adapter au processus de raffinement entre les échelles et au niveau de la même échelle. Le système de communication doit faire face aux imprévus ou à l'évolution des besoins des utilisateurs. Pendant la mission, les étapes d'exécution de l'activité sont: la phase d'exploration du champ d'investigation pour la localisation et l'identification de la situation d'urgence et la phase d'action après la découverte d'une situation critique et l'identification des événements.

Nous définissons les différents rôles des participants: Le superviseur de la mission, les coordinateurs et les investigateurs sur le terrain. Chaque groupe d'investigateurs est supervisé par un coordinateur. Chaque participant est associé à un identifiant, à un rôle et aux différentes fonctions qu'il remplit.

Ainsi, les participants à une OIU sont les suivants : Un superviseur qui gère l'ensemble des coordinateurs supervise toute l'application et attend des rapports de tous ses coordinateurs. Plusieurs coordinateurs dont chacun d'entre eux dirige une section d'investigateurs. Leurs fonctions est de collecter, interpréter et synthétiser les informations reçues des investigateurs et les diffuser vers le superviseur. Plusieurs sections d'investigateurs qui agissent pour aider, sauver et réparer. Ils explorent le champ opérationnel, observent, analysent et font un rapport décrivant la situation.

Nous illustrons dans la figure 6 un superviseur, trois types de coordinateurs qui sont le coordinateur de robots, le coordinateur d'avions, et le coordinateur de pompiers et trois types d'investigateurs qui sont des groupes de robots, d'avions

et de pompiers.

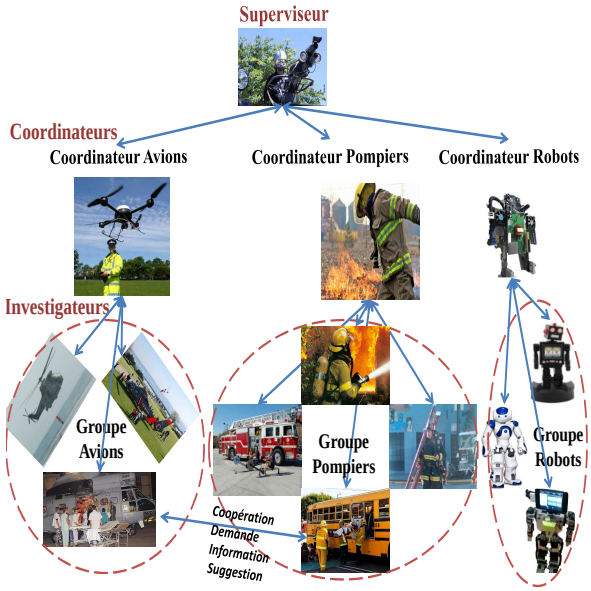

Figure 6: Étude de cas d'une OIU

Pendant la mission, les communications entre les participants est un élément clé pour la collaboration. La gestion des communications permet de s'assurer du bon déroulement de la mission.

Dans le scénario ERCMS, on distingue deux types de flux : le flux de coordination et le flux de coopération (Informations, demandes, suggestions). Le flux de coordination est entre les investigateurs et leur coordinateur correspondant ainsi qu'entre les coordinateurs et le superviseur. Les investigateurs transmettent des informations de coordination pour le coordinateur correspondant. Les participants doivent observer le champ d'investigation et rapporter sur ce qui est observé. Le flux de coopération est soit entre les investigateurs du même groupe (par exemple : entre 2 robots ou 2 pompiers) soit entre les investigateurs de différents groupes (par exemple : entre un pompier et un avion).

En s'appuyant sur cette étude de cas, nous procédons le passage du niveau style au niveau instance à travers la figure 7.

A l'échelle 0, nous associons l'instance d'une application de gestion d'intervention d'urgence (OIU). Nous présentons également les instances de ces deux points de vue de l'échelle 1. En effet, les trois participants d'une équipe d'Intervention d'Urgence sont représentés selon le premier point de vue de l'échelle 1 par les trois types composants qui sont le superviseur, le coordinateur et l'investigateur. Ces trois participants communiquent entre eux. Le deuxième point de vue présente ces relations sous forme d'associations à travers les ports. Nous illustrons également les instances de ces deux points de vue. En effet, un superviseur gère un ensemble de coordinateurs et chaque coordinateur dirige une section d'investigateurs. Ainsi, le point de vue 1 de l'échelle 2 représente trois composants composites de type superviseur, coordinateur et investigateur composant les trois participants de l'application des OIU. Ainsi, le point de vue 2 de l'échelle 2 illustre les instances pour chaque type de composants composites ainsi que les relations entre eux notamment les connexions entre les investigateurs du même groupe et entre les investigateurs de différents groupes (Flux de coopération).

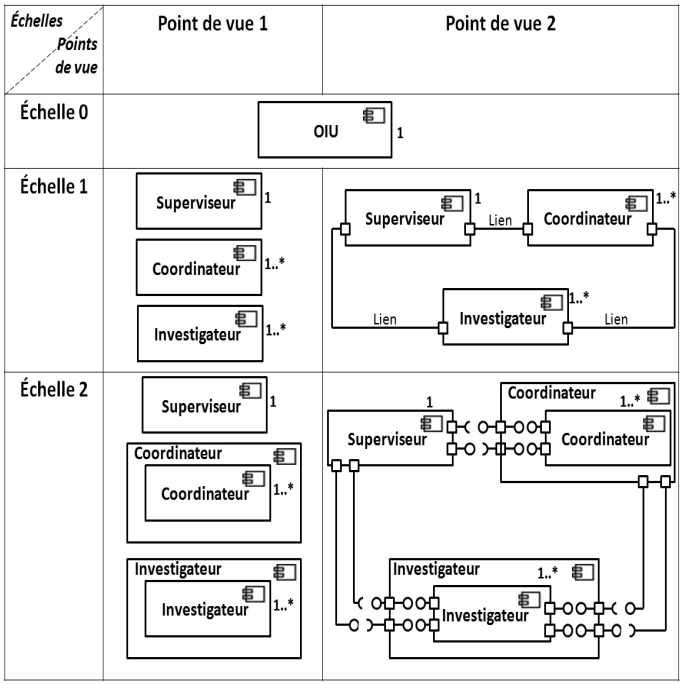

Figure 7: Illustration de l'approche

4. IMPLÉMENTATION ECLIPSE

Afin d'expérimenter et d'implémenter notre approche, nous avons développé une interface d'un plug-in conçu pour la modélisation d'un style architectural sous Eclipse[3]. Pour la réalisation de notre plug-in, nous nous sommes basées sur les frameworks GEF (Graphical Editing Framework)[4], EMF (Eclipse Modeling Framework)[2], GMF (Graphical Modeling Framework)[1]. GMF est un projet Eclipse avec la capacité de devenir le framework clé pour le développement rapide des éditeurs de modélisation graphique standardisés. Il fournit une composante de génération et d'exécution des infrastructures ce qui permet le développement rapide des éditeurs graphiques basés sur les frameworks EMF et GEF. Notre plug-in permet de créer un diagramme éditeur pour la conception d'un style architectural. Pour dessiner un élément graphique, il suffit de le tirer de la palette d'outils et de le mettre dans la partie éditeur de diagramme selon la technologie drag-and-drop. Nous allons ainsi, créer une instance de l'application, telle que le ferait un de ses futurs utilisateurs : nous allons, dans le cas de notre exemple, créer une instance de l'application de gestion des OIU que nous avons déjà détaillée.

A l'échelle 0, nous avons intégré un seul composant OIU (figure 8)

A l'échelle 1, nous avons présenté les 3 composants : Superviseur, Coordinateur et Investigateur. Chacun d'entre eux possède un ensemble de ports et d'interfaces Ainsi, nous

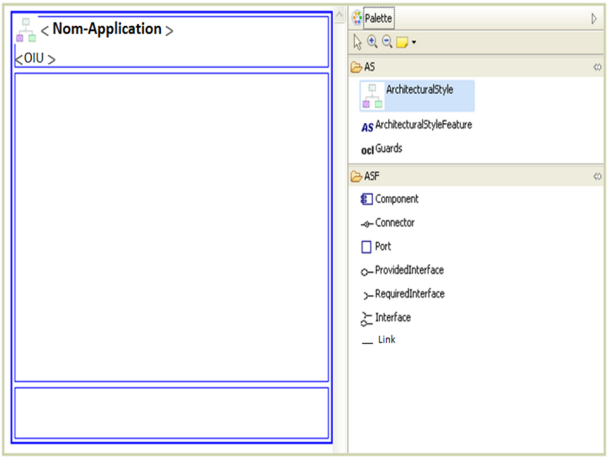

Figure 8: Représentation de l'échelle 0 dans le plug-in Eclipse

avons établi les relations entre eux à travers les liens.

A l'échelle 2, nous avons présenté notre style `Nom-Application` OIU avec les trois types de composants comprenant chacun un composant composite selon le premier point de vue puis nous avons ajouté les connexions établies entre eux selon le deuxième point de vue de cette échelle considérée.

Ainsi, notre plug-in Eclipse nous permet de représenter graphiquement notre approche de description des architectures multi-échelles et multi points de vue (figure 9) Notre plug-in est exporté sous forme des fichiers archives Jar. Ces fichiers seront intégrés avec l'installation standard d'Eclipse. Le concepteur peut l'utiliser en l'intégrant dans Eclipse avec ses dépendances. Le plug-in sera donc intégré dans l'interface de l'Eclipse. Le concepteur peut maintenant décrire son architecture par l'éditeur fourni.

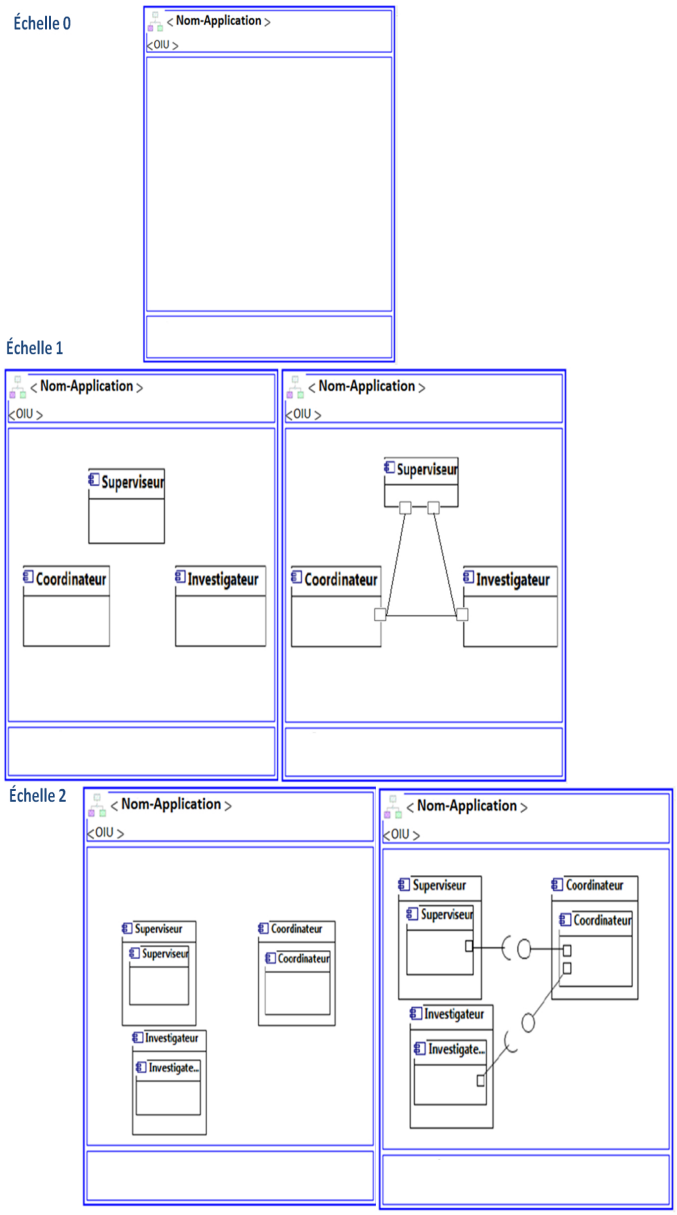

Figure 9: Implémentation

5. CONCLUSION ET PERSPECTIVES

Dans le cadre des travaux issus des domaines des architectures dynamiques, nous avons passé en revue l'état de l'art et nous avons étudié la littérature concernant les domaines de recherche abordés. Nous avons spécifié, en effet, quelques approches de modélisation des architectures multi-niveaux, multicouches, multi-vues et multi-styles ainsi que quelques méthodes de raffinement des architectures.

Ainsi, nous avons présenté au niveau conceptuel notre approche de description multi-échelles et multi points de vue pour les architectures logicielles dynamiques et nous avons proposé des notations visuelles, par extension des notations UML, pour les échelles et les points de vue au niveau du style de l'architecture.

Finalement, nous avons présenté le niveau d'instances sous un plug-in Eclipse qui implémente notre approche et la valide à travers une étude de cas détaillée des OIU. En effet, le développement de notre outil Eclipse a démontré que l'ajout de nouvelles fonctionnalités était souvent possible une fois l'apprentissage des cadres de développement Eclipse et GEF complétés. Nous n'avons eu qu'un seul problème de performance sérieux, et il était lié à la reconstruction trop fréquente des propriétés lorsque nous modifions des éléments. Ceci ralentissait parfois l'avancement du travail.

Nous travaillons actuellement sur l'amélioration de la démarche de validation de notre approche en se basant sur une correction notationnelle à travers le langage XML et une correction sémantique à travers le langage de transformation XSLT (eXtensible Stylesheet Language Transformations). Nous prévoyons dans un futur proche d'améliorer ce travail à travers la description des opérations de reconfigurations dynamiques et de contrôler que les modèles générés, représentés dans des descriptions XML, seront conformes à leurs méta-modèles. Nous envisageons de généraliser notre approche en considérant un nombre d'échelles quelconque et de spécifier également les notations et les règles génériques communes à toutes les échelles avec des lois de correspondance entre les échelles.

References

[1] http ://www.eclipse.org/gmf/. The eclipse foundation. graphical modeling framework.

[2] http ://www.eclipse.org/modeling/emf/. The eclipse foundation eclipse modeling framework.

[3] http://www.eclipse.org/. Site officiel de l'environnement eclipse.

[4] http://www.eclipse.org/articles/article-gef-emf/gef-emf.html. Using gef with emf.

[5] L. Baresi, R. Heckel, S. Thone, and D. Varro. Style-based refinement of dynamic software architectures. *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*, 2004.

[6] I. Bouassida. *Gestion dynamique des architecures logicielles pour les systèmes communicants collaboratifs*. PhD thesis, INSA, Toulouse, Février 2011.

[7] I. Bouassida, K. Drira, and J. Lacouture. Semantic driven self-adaptation of communications applied to ercms. In *24th IEEE International Conference on Advanced Information Networking and Applications*, 2010.

[8] I. Bouassida, K. Guennoun, K. Drira, C. Chassot, and M. Jmaiel. A rule-driven approach for architectural self adaptation in collaborative activities using graph grammars. *International Journal of Autonomic Computing*, 2010.

[9] B. Combemale, L. Broto, A. Tchana, and D. Hagimont. Metamodeling autonomic system management policies ongoing works. *IEEE International Workshop on Model-Driven Development of Autonomic Systems, Turku, Finland*, 2008.

[10] D. Garlan and M. Shaw. An introduction to software architecture. Technical report, CMU Software Engineering ng Institut, 1994.

[11] D. Hagimont, L. Broto, P. Stolf, N. Depalma, and S. Temate. Autonomic management policy specification in tune. *IEEE International Workshop on Model-Driven Development of Autonomic Systems*, 2008.

[12] Z. Juan, B. Xiaojuan, L. Qiang, C. Jie, and W. di. A component-based method for software architecture refinement. In *Proceedings of the 29th Chinese Control Conference*, 2010.

[13] G. Jung and J. Hatcliff. A type-centric framework for specifying heterogeneous, large-scale, component-oriented, architectures. *Science of Computer Programming*, 2009.

[14] S. kallel, M. HajdKacem, and M. Jmaiel. Modeling and enforcing invariants of dynamic software architectures. *Software and Systems Modeling*, 2010.

[15] khalil Drira. Reconfiguration models for adaptive and ubiquitous communication in layered services. Technical report, LAAS, Toulouse, 2009.

[16] S. Lange and M. Middendorf. On the design of two-level reconfigurable architectures. *IEEE Proceedings of the 2005 international conference on reconfigurable computing and FPGAs*, 2005.

[17] F. Mallet, C. Andre, and F. Lagarde. Un profil uml pour la modélisation multiniveau. *Technique et Science Informatiques (TSI)*, 2010.

[18] P. Muller and N. Gaertner. *Modélisation objet avec UML*. Eyrolles, 2004.

[19] M. Nassar, B. Coulette, J. Guiochet, S. Ebersold, B. ElAsri, X. Cregut, and A. Kriouile. Vers un profil uml pour la conception de composants multivues. *L'OBJET*, 2004.

[20] F. Oquendo. π-method: a model-driven formal method for architecture-centric software engineering. *ACM SIGSOFT Software Engineering Notes*, 2006.

[21] L. Piechocki. *UML, le langage de modélisation objet unifié*. developpez.com, 2007.

[22] V. Rafe, M. Reza, Z. Miralvand, R. Rafeh, and M. Hajiee. Automatic refinement of platform independent models. *IEEE International conference on computer technology and development*, 2009.

[23] G. Sancho. Modélisation multi-niveau pour des systèmes ubiquitaires collaboratifs. *Congrés de doctorants*, mars 2009.