Approche de déploiement automatique basé sur les modèles sémantiques

Hatem Arous, Jorge Ricardo Gomez Montalvo, Thierry Villemur, Ernesto Expósito

▶ **To cite this version:**

Hatem Arous, Jorge Ricardo Gomez Montalvo, Thierry Villemur, Ernesto Expósito. Approche de déploiement automatique basé sur les modèles sémantiques. 6ème Conférence francophone sur les architectures logicielles (CAL 2012), May 2012, Montpellier France. 11p. hal-01477159

HAL Id: hal-01477159
https://laas.hal.science/hal-01477159

Submitted on 27 Feb 2017

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.

Approche de déploiement automatique basé sur les modèles sémantiques

Hatem Arous[1,2] Jorge Gómez-Montalvo[1,2] Thierry Villemur[1,2] Ernesto Exposito[1,2]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France

[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France

{hatem.arous, jorge.gomez, villemur, ernesto.exposito} @laas.fr

RESUME

Le domaine des systèmes distribués est en pleine mutation avec l'avènement des dispositifs légers et très mobiles au sein d'une informatique omniprésente. Les principaux défis de tels systèmes incluent la gestion des communications et des données, le déploiement des entités échangeant des données et la gestion des paramètres de la qualité de service. La gestion des communications et des échanges de données sont actuellement intégrées à l'intérieur de la couche middleware. Cependant le déploiement des composants logiciels et la gestion des paramètres de la qualité de service (configuration et reconfiguration lors de l'exécution) restent des défis à relever et à intégrer dans la couche middleware.

Dans cet article, nous présentons une approche de déploiement automatique d'entités logicielles de communications pour les systèmes distribués. Cette approche repose sur un modèle sémantique (ontologie) conçu pour modéliser la structure distribuée des systèmes de communications. Le modèle proposé prend en compte la configuration initiale des paramètres de la qualité de service et de leurs valeurs au niveau de toutes les entités communicantes ainsi que la reconfiguration de ces paramètres en cours d'exécution.

Mots clés

Déploiement automatique, génération automatique de code, MDA, ODA, qualité de service (QdS)

1. INTRODUCTION

Les applications distribuées coopératives multiutilisateurs et multiservices manipulent différents types de médias et impliquent des composants et des services diffus interagissant selon des schémas de coopération composés en temps réel.

Dans le futur, l'information sera diffusable beaucoup plus largement par une utilisation massive d'équipements embarqués : les systèmes de communication seront ubiquistes, mobiles et hétérogènes, autorisant la mise en œuvre d'activités coopératives complexes dans des environnements peu ou pas pré-équipés en infrastructure réseau, typiquement sans fil et ad-hoc.

Dans ce contexte, un enjeu de recherche majeur est de permettre aux applications et à leurs utilisateurs de disposer de services de coopération et de communication de groupe qui offrent des qualités de service (QdS) adaptées, en fonction des ressources disponibles, aux capacités hétérogènes et variables.

Ces systèmes de communication coopératifs et distribués sont très complexes à concevoir et à gérer lors de l'exécution vue leur dynamique au niveau de la topologie ou au niveau de la gestion des communications.

Les solutions envisagées sont proposées en deux niveaux ;

- Au niveau de conception et de l'implémentation, simplifier la conception des systèmes distribués en se basant sur des modèles aidera les concepteurs à les concevoir et même à automatiser pas la suite le développement des codes source des entités communicantes.
- Au niveau est l'exécution, deux solutions se présentent : le déploiement dynamique des entités et la reconfiguration dynamique des paramètres de QdS.

Les technologies du web sémantique semblent être prometteuses pour cet effet. En effet, plusieurs initiatives sont apparues pour annoter sémantiquement les services Web, fournissant diverses descriptions qui peuvent être utilisés pour décrire le contexte de la communication coopérative adaptative [11]. Ces descriptions sont généralement basées sur des ontologies.

Nous proposons un modèle sémantique pour assurer la description des systèmes distribués complexes en vue d'assurer leur auto-adaptation aux changements de contexte. Nous avons développé une approche guidée par les ontologies pour la conception de systèmes de communications distribués. Cette approche vise à faciliter la conception, le développement et le déploiement des entités middleware de communication avec une configuration initiale des paramètres de QdS. Une fois déployées, les entités de communication gèrent les flux de communication entre les applications coopératives distribuées.

Les modules logiciels nécessaires pour la chaîne de déploiement ont été testés sur le middleware Data Service Distribution [3].

Cet article est organisé comme suit :

La section deux présente la motivation de notre travail et le positionne par rapport à quelques travaux déjà faits pour résoudre le problème de déploiement dans les systèmes complexes.

La troisième section décrit l'architecture de la chaîne de déploiement proposée dans le cadre de notre approche qui repose sur les modèles sémantiques et suit l'approche Ontology Driven Architecture (ODA).

La quatrième section présentera l'ontologie middleware que nous avons conçue pour modéliser les systèmes de communications distribués qui utilisent une couche middleware.

La cinquième section sera consacrée pour décrire les réalisations. Elle présentera le middleware utilisé et les modules développés dans la chaîne de déploiement.

Nous concluons par un bilan des travaux réalisés et les travaux futurs.

2. MOTIVATION ET POSITIONNEMENT

2.1 Définition du déploiement

Le déploiement est le processus de distribution des composants logiciels sur les nœuds des utilisateurs en prenant en compte les informations contextuelles comme le rôle de l'utilisateur, les caractéristiques des appareils et tout autre élément qui pourrait influencer l'opération de déploiement [13]. Le processus de déploiement doit garantir la bonne exécution des applications logicielles distribuées qui supportent les activités coopératives des groupes d'utilisateurs [11]. Ce processus peut se faire manuellement par un utilisateur privilégié. Malheureusement, le déploiement manuel est une tâche compliquée, consommatrice de temps et source d'erreurs.

Le déploiement automatique des composants éliminera les inconvénients précédents mais sa difficulté majeure serait de déduire et de gérer, d'une manière automatique, des paramètres fonctionnels et non-fonctionnels (fiabilité, coût des opérations, QdS…) provenant des exigences de plus haut niveau de la coopération des utilisateurs et de leurs applications communicantes.

Dans la même direction de travaux faits par Bouassida et Drira [2], et Villemur et Sancho [13,11], nous proposons de supporter la capture et la gestion de ces paramètres en utilisant une approche guidée par les modèles. Un autre travail fait par Exposito et Gómerz-Montalvo [4] propose une approche basée sur les modèles et qui suit l'approche guidée par les ontologies. Cette approche a produit un Framework autonome de gestion de QdS dans les réseaux domestiques.

2.2 Travaux connexes

Les services de déploiement dynamiques des composants informatiques sont très utiles dans les systèmes de communication distribués à architectures complexes tels que les grilles informatiques ou l'informatique en nuage (cloud computing). En effet ils permettent d'économiser le temps de déploiement, de réduire les coûts et surtout de réduire les taux d'erreurs induits par le déploiement manuel.

Dans D. Xiaoshe et al. [15], les auteurs présentent un processus pour le déploiement automatique des services de surveillance ADMS dans les grilles informatiques. La solution proposée a été testée sur différentes grilles telles que Dawing4000 et RS6000, et différents systèmes d'exploitation tels que Red hat Linux et AIX. Le seul paramètre à tester est le temps de déploiement du service de surveillance qui est nettement inférieur au temps de déploiement des grappes de serveurs (clusters), et aussi nettement inférieur au temps de déploiement d'autres systèmes de surveillance. Cependant ce service de déploiement ne s'intéresse qu'au temps de déploiement mais il n'y a pas de critères non-fonctionnels (QdS, fiabilité…) pris en charge.

L'approche présentée dans [7] vise à déployer des services en se basant sur une politique de QdS. Elle traite quatre paramètres de QdS : Disponibilité, Fiabilité, Délai et Capacité. Cette approche est proche de la nôtre car elle traite le déploiement des services en se basant sur les aspects fonctionnels et non-fonctionnels. Néanmoins, la solution que nous proposons est plus générique. En effet, en modélisant les paramètres de QdS selon le standard ITU-T X.641, nous pouvons définir plusieurs propriétés pour chaque paramètre de QdS et non seulement trois (la valeur à la conception, la valeur actuelle et la valeur du seuil maximum ou la limite de la ressource pour un fonctionnement normal).

3. APPROACHE ET ARCHITECTURE

Notre proposition est conçue en suivant l'approche ODA (Architecture Dirigée par les Ontologies ou en anglais Ontology Driven Architecture) qui raffine le Framework MDA (Architecture Dirigée par les Modèles ou en anglais Model Driven Architecture).

3.1 Architecture Dirigée par les Ontologies

L'approche ODA a été proposée par le World Wide Web Consortium (W3C) dans le but de promouvoir l'utilisation des modèles sémantiques ou les ontologies dans le framework MDA [8, 9]. L'utilisation des technologies du Web sémantique est destinée naturellement à étendre le framework MDA en définissant sans ambiguïté les vocabulaires du domaine et en fournissant la vérification et la validation de la cohérence du modèle [1]. Ces technologies sont basées principalement des langages de spécifications tels que RDF et OWL.

Les ontologies peuvent être utilisées pour représenter des services permettant à des fonctionnalités déclaratives d'être déployées, découvertes et réutilisées. Leur utilisation facilite le développement de logiciels en permettant la découverte et la composition de fonctions existantes pour fournir une nouvelle fonctionnalité plutôt que de construire une nouvelle solution.

ODA peut être utilisée pour construire des modèles sémantiques qui supportent la spécification et la conception. Ces modèles peuvent être intégrés dans l'implémentation du système en incluant l'identification et les descriptions des composants et permettant ainsi leur découverte et leur réutilisation lors de la conception et de l'exécution. Ces modèles basés sur les ontologies capturent la sémantique liée aux propriétés, aux relations et au comportement des composants. Comme l'ontologie est un modèle conceptuel explicite avec une sémantique formelle basée sur la logique, les descriptions des composants peuvent être interrogées ou vérifiées pour éviter les compositions incompatibles.

3.2 Architecture de la chaîne de déploiement

L'architecture de notre chaîne de déploiement est proposée dans le cadre de l'Architecture Dirigée par les Ontologies. L'utilisation de l'approche ODA permet de simplifier la conception et la modélisation des systèmes. Elle offre également la possibilité de vérifier la cohérence du modèle de système avant d'implémenter ses différentes parties. En outre, cette approche simplifie la génération automatique du code source. Dans notre travail, ODA permet de représenter l'architecture de notre système pour un déploiement automatique ultérieur et aussi d'exprimer les exigences non-fonctionnelles du système dès la conception.

La figure 1 présente l'architecture de la chaîne de déploiement qui consiste en trois modules de base : un module de modélisation du système, un module de génération automatique de code, et un module de déploiement automatique. Les trois modules sont activés en séquence. Leurs rôles sont détaillés dans les sections qui suivent.

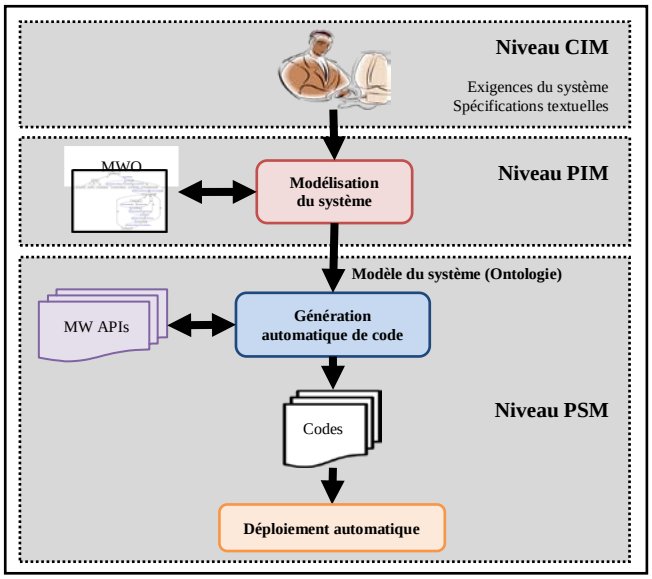

Figure1: Architecture de la chaîne de déploiement

3.3 Module de modélisation du système

Ce module gère la conception d'un système de communication réparti. Pour concevoir le modèle sémantique et pour modéliser tout système distribué, nous suivons l'approche ODA, une spécialisation du Framework MDA, basée sur le concept de l'ontologie.

Le modèle sémantique est composé d'un ensemble d'ontologies que nous appelons Ontologie Middleware (MWO). Cette ontologie fournit les bases pour l'utilisateur de notre approche pour concevoir un système de communication distribué. En utilisant MWO, un concepteur peut générer une instance qui décrit les entités qui participent à la communication, leur configuration initiale des paramètres de QdS et la topologie globale du système.

Deux étapes composent ce module. La première étape prend en compte les exigences de haut niveau, la seconde s'intéresse aux exigences de communication de bas niveau.

3.3.1 Modélisation de QdS

Cette étape considère le comportement non-fonctionnel du système qui est présenté par la configuration des paramètres de QdS de chaque entité de communication. L'architecture du système et le comportement non-fonctionnel seront décrit en utilisant un modèle sémantique basé sur le concept d'ontologie. Ainsi nous pouvons considérer les comportements fonctionnels et non-fonctionnels dès la conception et aussi lors de l'exécution.

3.3.2 Modélisation de la topologie du système

Dans les systèmes distribués, un middleware est une couche logicielle qui établit, maintient et facilite les communications entre les entités du système de communication. Dans notre travail, nous considérons la modélisation de la topologie du système au dessus des couches middleware.

Le module de modélisation du système contient un ensemble d'ontologies prédéfinies qui se réfèrent à plusieurs profils de middlewares. Par exemple, les middlewares adoptant le modèle architectural publier/souscrire tels que DDS, JMS et AMQ, les middlewares orientés objet tel que CORBA...

Durant cette deuxième étape, le concepteur choisit un profil de middleware et étend l'ontologie MWO. Cette extension permet de prendre en compte la topologie de l'ensemble du système.

A la fin des deux étapes, l'instance du MWO modélise à la fois les paramètres de QdS et la topologie du système de communication utilisant un middleware choisi.

3.4 Module de génération automatique de code

Ce module génère le code source des entités de communication. Cette création est basée sur le modèle de communication sélectionné dans l'ontologie par le concepteur du système.

Ce module nécessite en entrée le choix d'un langage de programmation spécifique (par exemple Java) et une plate-forme middleware spécifique en conformité avec le paradigme de communication middleware inclus dans l'ontologie MWO.

La génération automatique de code requiert deux étapes :

- Identifier les exigences fonctionnelles et non-fonctionnelles de chaque entité logicielle,
- Générer le code source de chaque entité.

3.5 Module de déploiement automatique

Ce module déploie les entités du système sur les nœuds du réseau en tenant compte de la politique de QdS requise qui est décrite dans le modèle du système. Il crée la correspondance entre la conception et le fonctionnement (exécution) de l'application. Pendant le processus de déploiement, les entités sont installées, configurées et activées sur la plateforme.

Ce processus est supervisé par une architecture à deux niveaux hiérarchiques. Le niveau le plus haut est composé d'une entité de supervision (Superviseur) qui contrôle l'ensemble des entités exécutrices (Exécuteur). Chaque nœud possède un exécuteur.

Le processus de déploiement fonctionne comme suit : le superviseur reçoit les codes des entités à déployer avec leurs caractéristiques de configuration requise .Ensuite, il transmet à chaque exécuteur les codes des entités qui seront sous son contrôle. Après, chaque exécuteur installe, configure et active les codes des entités sous sa responsabilité.

4. ONTOLOGIE MIDLEWARE (MWO)

L'ontologie MWO proposée est une contribution centrale de notre travail. Cette section détaille ses principales caractéristiques. Elle présente comment les exigences de haut niveau définies par les paramètres de QdS sont capturées et modélisées et comment les exigences de bas niveau, définies par le choix du middleware de communication, sont prises en compte.

4.1 Caractéristiques de l'ontologie MWO

La figure 2 présente une partie de la structure de MWO qui décrit les middlewares basé sur le modèle publier/souscrire, conçue avec l'éditeur d'ontologies «Protégé». MWO est principalement composée de quatre classes: la classe principale « Middleware », la classe «Entity», la classe «QoSPolicy», et la classe «ITU-T_X.641». MWO peut être raffinée par les technologies middleware existantes qui apparaissent sur la figure 2 dans les cercles pointillés reliés au concept Middleware.

La classe «Entity» représente les composants logiciels qui seront déployés dans le réseau. Cette classe représente une ontologie que nous avons développée et qui regroupe l'ensemble des entités communicantes que nous pourrons utiliser dans un système

communicant. Une entité décrite dans cette ontologie peut être utilisée avec plusieurs middlewares adoptant le même modèle de communication. Comme chaque entité a ses propres paramètres de QdS, et puisque certains paramètres de la QdS peuvent être appliqués à une ou plusieurs entités, nous avons implanté une ontologie «QoSPolicy» qui inclut tous les paramètres de QdS que nous gérons. Pour décrire les paramètres de QdS, nous utilisons l'ontologie de l'ITU-T X.641 développée dans [4, 5]. L'ontologie «QoSPolicy» regroupe l'ensemble des paramètres décrits. Les exigences raffinées apparaissent sur la Figure 2 avec des cercles en pointillés connectés au concept de politique de QdS. Ils seront détaillés dans la section suivante.

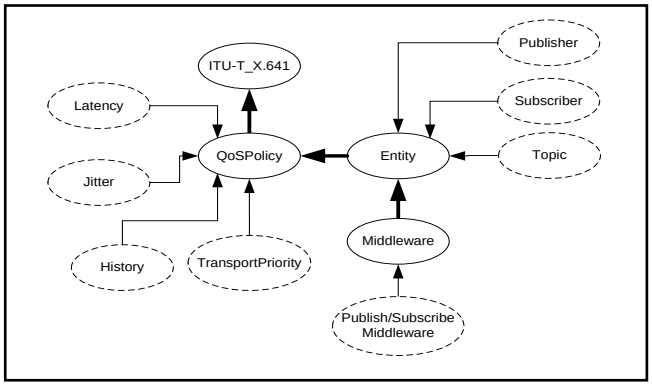

Figure 2 : Représentation partielle de l'Ontologie Middleware

4.2 Concept de la Qualité de service

Le concept de QdS est introduit dans les différents domaines d'échange de données tels que les réseaux de capteurs sans fil, les réseaux mobiles ad-hoc, les systèmes distribués et l'informatique en nuage [14]. Dans tous ces domaines, le terme «qualité de service» fait référence à des mécanismes de réservation et de contrôle de ressources.

La QdS est définie par un ensemble de paramètres qui réagissent les communications. Elle est également utilisée pour gérer les priorités entre les différentes applications, les utilisateurs, les flux de données, ou pour maintenir et garantir un certain niveau de performance au sein des échanges de flux de données. Garantir la QdS est un enjeu majeur des systèmes informatiques actuels.

Les principaux paramètres de QdS identifiés sont :

- Durabilité : Elle indique si les données doivent être conservées ou non après leur livraison.
- Temps de latence : Il définit le délai maximum acceptable entre la date d'envoi de données et la date de notification de la réception de données.
- Fiabilité : Elle définit la période d'activité fiable du système.
- Gigue : Elle mesure la variation du délai requis par un paquet entre sa source d'émission et sa destination de réception. Toute valeur élevée de la gigue peut affecter sérieusement la qualité des flux de données.
- Priorité: Lorsque plusieurs services ou applications partagent des ressources limitées communes, les priorités peuvent être fixées à ces services ou applications pour gérer les conflits.

4.3 Concept de la plateforme middleware

Une plateforme middleware est une couche logicielle incluse entre la couche du système d'exploitation et la couche application. Elle crée une couche uniforme de communication pour échanger des informations entre différentes applications. Ces applications sont connectées à la plateforme middleware à travers des interfaces logicielles fournies par cette dernière.

Les composants logiciels des middlewares assurent la communication indépendamment des caractéristiques matérielles et logicielles du système hôte final et des composants du réseau, plus précisément les protocoles réseau et les systèmes d'exploitation, impliqués.

Les plateformes middleware peuvent être classées comme présenté dans [6] selon:

- La topologie du réseau : middleware centralisé (Client / Serveur) vise à vis de middlewares distribués (Peer-to-Peer).
- La granularité de la communication : message, paramètres de sous-programmes et objet.
- Le paradigme de communication : Point-à-Point, multipoint, ou publier / souscrire.

Dans notre travail, nous avons choisi de modéliser les plateformes middleware en conformité avec le paradigme de la communication qu'elles respectent. En effet, le paradigme de communication influence fortement la manière avec laquelle l'application se connecte et utilise l'interface middleware fournie. Comme MWO est reliée à la distribution des composants de l'application, ce choix nous semble le plus logique et pertinent.

La figure 2 montre une partie de MWO qui décrit les middlewares adoptant le paradigme publier/souscrire. Les concepts apparaissent dans les cercles en pointillés (Publisher, Subscriber et Topic pour les entités…)

5. REALISATION DE LA CHAINE DE DEPLOIEMENT

Notre première réalisation a été faite en considérant les middlewares publier/souscrire. En effet, certains d'entre eux intègrent la notion de la QdS. Cette propriété nous semble utile pour la construction des applications basées sur la QdS.

5.1 Middlewares Publier/Souscrire

5.1.1 Présentation

Plusieurs middlewares publier/souscrire intègrent des propriétés avancées telles que la performance, le coût, l'évolutivité, la QdS, la robustesse... Parmi eux nous citons JMS et DDS.

Dans les middlewares publier/souscrire, les messages échangés sont associés à un sujet (Topic). Un Topic est une étiquette qui identifie un domaine d'intérêt spécifique. Les consommateurs s'abonnent aux sujets avec lesquels ils ont une certaine affinité. Une fois l'abonnement effectué, tout message associé à ce Topic sera automatiquement reçu par le groupe d'abonnés. Un message est supprimé de la file du middleware lorsque tous les abonnés l'ont lu et l'ont acquitté (ou si sa date d'expiration est atteinte).

5.1.2 Middleware Data Distribution Services

DDS [3] est un middleware orienté message, distribué, qui opère selon le modèle de communication publier/souscrire. C'est une plateforme normalisée par l'Object Management Group (OMG). Elle permet aux applications faiblement couplées d'échanger des informations indépendamment de l'architecture sous-jacente, des langages de programmation ou des systèmes d'exploitation [10].

Ce middleware est destiné principalement aux applications industrielles distribuées nécessitant de fortes contraintes en termes de fiabilité et de performance, dans des domaines tels que l'aéronautique ou la défense. Il fournit des services avancés

comme le service de découverte automatique et facilitent la tâche de l'utilisateur pour la gestion des communications.

Les spécifications de DDS décrivent trois couches, présentées ci-après du bas vers le haut:

- Le protocole Real-Time Publish/Subscribe (RTPS) [12] : conçu pour être capable de s'exécuter au-dessus des protocoles de transport multidiffusion (multicast) et best-effort comme UDP/IP.
- La couche Data-Centric Publish/Subscribe (DCPS) [3], qui garantit la transmission avec QdS garantie des données (c-à-d transmettre la bonne information à la bonne destination).
- Une couche supérieure optionnelle nommée Local Data Reconstruction Layer (DLRL), qui renforce l'intégration de DDS dans la couche application.

Le principe de fonctionnement s'appuie sur un mécanisme pour la publication et la souscription, permettant aux applications de :

- Modifier les données partagées et par la suite notifier les autres applications par la publication,
- Souscrire aux données partagées et recevoir les modifications apportées par les applications distantes.

Le choix du middleware DDS est motivé par la flexibilité de déploiement des services et par la facilité d'utilisation du service de découverte automatique des nouvelles entités DDS et aussi par la possibilité de reconfiguration des paramètres de la QdS qui influencent le comportement des services de DDS. Ces paramètres peuvent être associés à chaque entité DDS [3].

5.2 Génération automatique de code

Les modules précédents de notre architecture de la chaîne de déploiement sont indépendants de la technologie des couches sous-jacentes, en conformité avec l'approche ODA. Par conséquent, les caractéristiques fines de DDS ne sont prises en compte que lors de la génération de code.

Pour générer automatiquement le code source des entités DDS décrites dans l'instance de l'ontologie du système, nous avons développé un Générateur Automatique de Code Source que nous appellerons par la suite «le générateur». L'entrée du générateur est une instance de l'ontologie qui décrit le système. Le fichier de MWO contient les descriptions des entités intermédiaires, les liens entre elles (par exemple, quel Publisher publie dans quel Topic...) et les descriptions des données échangées. Ce fichier contient aussi une configuration initiale des paramètres de QdS. Cette configuration définit la valeur de chaque paramètre pour chaque entité. Toutes ces données sont utilisées pour générer les codes source de ces entités communicantes.

Le générateur est composé de deux parties principales : un analyseur d'ontologie, plus précisément de fichiers OWL, et un moteur responsable de la génération du code source.

Quand le fichier est présenté au générateur, l'analyseur extrait l'architecture du système et identifie les entités communicantes, les liens entre elles, leurs propriétés, la configuration des paramètres de QdS de chaque entité et leurs valeurs initiales. Ensuite, le moteur du générateur de code génère les codes source des entités dans le langage de programmation approprié. Ce code peut fonctionner sur la distribution choisie de DDS. En effet, de nombreuses implémentations telles que RTIDDS, OpenSplice DDS et OpenDDS peuvent être trouvées en conformité avec les spécifications OMG DDS. Certaines implémentations ont des APIs différentes selon différents langages de programmation ciblés tels que C, C++, C# ou Java. Le choix de la distribution DDS (ou implémentation) et le langage de programmation peuvent être fait lors de la conception du système (dans l'instance de l'ontologie du système) ou lors de la génération du code source via l'interface graphique du générateur.

5.3 Déploiement automatique

Le déploiement automatique de composants logiciels traite du problème de l'installation et de la configuration de cess composants dans des environnements complexes et hétérogènes tels que les systèmes de communication distribués. Un service de déploiement automatique doit considérer les exigences fonctionnelles et non-fonctionnelles du système.

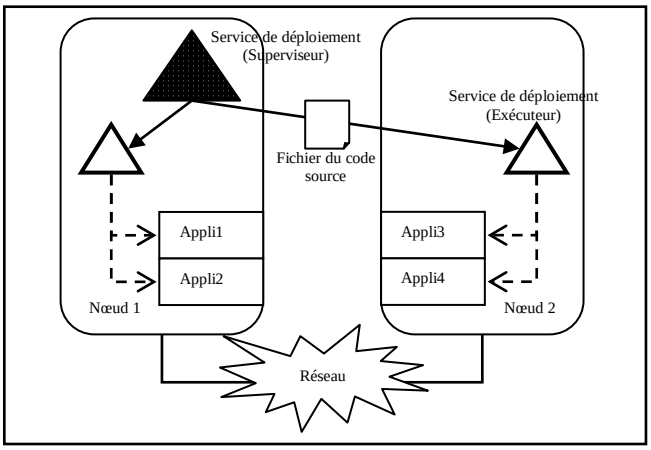

Figure 3: Architecture hiérarchique du service de déploiement

Dans notre approche, nous traitons du déploiement des composants responsables de la communication dans le système distribué. Nous optons pour un service de déploiement distribué : chaque nœud du réseau possède un service de déploiement local appelé «exécuteur» (représenté dans la figure 3 par un triangle blanc) responsable du déploiement des composants logiciels qui s'exécutent sur ce nœud. Cet exécuteur recevra les codes des composants dans le but de les installer localement. Comme la configuration initiale des paramètres de QdS de chaque entité est implémentée dans le code source, l'exécuteur garantit que la politique initiale de QdS est bien respectée lors du déploiement.

Puisque le générateur ne peut que générer du code source, une entité principale, qui envoie le code source de chaque entité à un exécuteur pour être installé et exécuté, est nécessaire. Nous avons donc développé un service de déploiement central que nous appelons «superviseur» (représenté dans la figure 3 par un triangle noir) qui agit en tant que coordinateur et qui distribue les codes, afin de les installer et exécuter par le bon exécuteur, basé sur l'architecture du système décrite dans l'instance de l'ontologie qui représente notre système de communication distribué.

6. ETUDE DE CAS : COVOITURAGE

Notre étude de cas est effectuée dans le cadre du projet AMIC-TCP (pour Architecture de Multiplexage Informatique et de Communication pour les Transports en Commun des Personnes) est un projet national français dirigé par la société ACTIA pour développer une architecture de communication unifiée reliant tous les modules électroniques au bord des bus et fournissant une passerelle pour les nouvelles.

Dans l'ensemble d'applications distribuées, le scénario coopératif suivant été proposé. Un passager équipé d'un appareil mobile

(téléphone intelligent, ordinateur portable…) prend un bus. Il peut se connecter au réseau de bus et accéder à une interface qui offre la sélection de plusieurs applications existantes et de les déployer sur son appareil. Dans notre cas, le passager est intéressé par une application de covoiturage. Elle présente de nombreuses annonces et un salon de discussion pour chaque annonce. Lors du déploiement de cette application localement, le service de déploiement crée un Subscriber qui permet de lire des annonces et dans le cas ou l'utilisateur choisit une annonce, un Publisher et un Subscriber seront créés pour assurer la discussion avec d'autres passagers intéressés dans le même bus ou dans un autre bus. Elle permet aussi de créer un nouveau Topic pour chaque nouvelle annonce au fur et à mesure qu'un passager le souhaite.

Dans de telles circonstances, nous devons faire face aux problèmes de ressources limitées au niveau du bus et aussi au niveau du matériel des. Dans ce cas nous avons un problème de stockage. De plus au niveau des appareils mobiles, leur source d'énergie, la batterie, est très limitée. Une solution possible pour résoudre ces problèmes, est d'éliminer les échanges de données inutiles qui pourront saturer la mémoire ou épuiser la batterie. De plus il faut faire des échanges très rapides et efficaces.

Pour résoudre ces problèmes nous avons défini une politique de QdS. En utilisant DDS cette politique se traduit par la manipulation de 2 paramètres de QdS :

- DURABILITY : qui opère au niveau du Topic, du DataWriter ou du DataReader [3]. Il permet de garder ou non une donnée dans le réseau quand il n'y pas de consommateur. Pour ne pas surcharger le réseau on peut sauvegarder la donnée uniquement au niveau de l'émetteur, et dans ce cas la valeur de ce paramètre sera « TRANSIENT_LOCAL ».
- HISTORY : ce paramètre permet de gérer le nombre de mises à jour des données échangées à sauvegarder pour qu'un éventuel consommateur se connecte et les consomme. Nous n'allons sauvegarder que la dernière mise à jour d'une donnée. Pour le faire, la valeur de ce paramètre est « KEEP_LAST ».

7. CONCLUSION

Dans cet article nous avons présenté une approche qui permet le déploiement des entités logicielles constituant un système de communication distribués. Notre approche est basée sur une Ontologie Middleware (MWO) qui permet de modéliser un système de communication distribué et d'attribuer une configuration initiale des paramètres de QdS à chacun de ses entités. Après la modélisation du système, un générateur automatique de code source génère les codes source des entités du système. Ensuite les entités du système peuvent être automatiquement déployées et activées sur le réseau.

Actuellement, nous sommes entrain de faire évoluer notre chaîne de déploiement.

Dans des travaux futurs nous visons à maintenir le bon fonctionnement du système en maintenant un accord sur une configuration des paramètres de QdS fait lors de l'exécution. Pour le faire, nous devons concevoir et développer deux services : un service de reconfiguration des paramètres de QdS et un service de surveillance (monitoring). Le service de reconfiguration des paramètres de QdS modifiera les valeurs des paramètres dynamiquement lors de l'exécution. Le service de surveillance veillera à ce que les bons paramètres de QdS soient mis à jour par les bonnes valeurs et avertira le service de reconfiguration si un problème survient lors de cette mise à jour.

8. REFERENCES

[1] Berners-Lee, T., Hendler, J., and Lassila, O. 2001. The Semantic Web, Scientific American , May, 2001.

[2] Bouassida, I., Drira, K., Chassot, C., and Jmaiel, M. 2010. "An architectural refinement model for group-wide communications with priorities applied to the rosace project senario", International Transactions on Systems Science and Applications (ITSSA), Vol.6, N°4, pp.339-349, November 2010.

[3] Data Distribution Service for Real-time Systems Version 1.2 OMG Available Specification formal/07-01-01.

[4] Gómez-Montalvo, J., Lamolle, M., and Exposito, E. 2009. "A Multimedia Ontology Driven Architecture framework (MODA) for networked multimedia systems," Networked Digital Technologies, 2009. NDT '09. First International Conference on , vol., no., pp.411-416, 28-31 July 2009.

[5] Gómez-Montalvo, J. and Exposito, E. 2010. "A Semantic Approach to User-based QoS Provision for Multimedia Services in Home Networks", Third International Conference on Communication Theory, Reliability, and Quality of Service, 2010.

[6] Ibrahim, N. 2009. "Orthogonal Classification of Middleware Technologies", Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009.

[7] Kessal, S., and Simoni, N. 2011. "A Deployment of Service Elements based on QoS", IEEE World Congress on Services, 2011.

[8] MDA Guide Version 1.0.1, OMG, June, 2003.

[9] Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering, W3C Working Draft, April, 2006.

[10] Ryll, M., and Ratchev, S. 2008. "Toward a publish/subscribe control architecture for precision assembly with the Data Distribution Service", IFIP 2008, Volume 260, pp.359-369, Micro-Assembly Technologies and Applications.

[11] Sancho, G., Villemur, T., and Tazi, S. 2010. "An ontology-driven approach for collaborative ubiquitous systems", International Journal of Autonomic Computing (IJAC), Vol.1, N°3, pp.263-279, May 2010.

[12] [12]The Real-time Publish/Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification, Version 2.1, OMG formal/2009-01-05.

[13] Villemur, T., and Hammami, E. 2008. "Design and evaluation of a context-aware service deployment for collaborative sessions", Computer Communications, Vol.31, N°17, pp.4176-4191, Nov. 2008.

[14] Xiao, Y., Lin, C., Jiang, Y., Chu, X., and Shen, X. 2010. "Reputation-based QoS Provisioning in Cloud Computing via Dirichlet Multinomial Model," IEEICCC 2010.

[15] Xiaoshe, D., Yinfeng, W., Zhongsheng, Q. and Fang, Z. 2006. "Research on an Automatic Deployment Mechanism of Monitor Service in Grid Environment", Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops (GCCW'06), IEEE, 2006.