Large scale interoperability in the context of Future Internet

Preston Rodrigues

N° d'ordre: 4786

THÈSE

présentée devant

L'UNIVERSITÉ DE BORDEAUX
École Doctorale de Mathmatiques et Informatique

pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX
Mention INFORMATIQUE

par

Preston Francisco RODRIGUES

Équipe d'accueil : PROGRESS
École Doctorale : Mathématiques et Informatique
Composante universitaire : LABRI

Titre de la thèse :

Large scale interoperability in the context of Future Internet

Interoperabilit large chelle dans le contexte de l'Internet du futur

à soutenir le 27 May 2013 devant la commission d'examen

MM. :	Mohamed	MOSBAH	Président
MM. :	Didier	DONSEZ	Rapporteurs
	Philippe	ROOSE	
MM. :	Gilles	MULLER	Examinateurs
	Gaël	THOMAS	
	Laurent	RÉVEILLÈRE	
	Daniel	NÉGRU	
	Yérom-David	BROMBERG	

to my parents,

Acknowledgement

First of all, I would like to thank Dr. Laurent Réveillère, my supervisor, who provided tremendous support and advice for the duration of my thesis work. His insight helped my work go in the right direction, and he taught me the principles of research work. I am extremely grateful for this. It gives me immense pleasure to acknowledge and convey my heart felt appreciation for the assistance and feedback given by Dr. Yérom-David Bromberg. His constant motivation and advise helped me improve my work. I am also grateful to Dr. Daniel Négru for his valuable feedback and for providing the necessary funds to complete my PhD. He was instrumental in exposing me to the working of an integrated European project.

My sincere thanks goes to Prof. Mohammed Mosbah, University of Bordeaux, who gracefully accepted to chair my defense committee. I would also like to thank Prof. Didier Donsez, University of Grenoble and Dr. Phillipe Roose, IUT de Bayonne who accepted to review my thesis document before the defense. Special thanks goes to Prof. Gilles Muller, Director of Research INRIA and Dr. Goël Thomas, UPMC – LIP6 for being part of the defense committee.

I would also like to thank CNRS-LaBRI for hosting me for the duration of my PhD. No amount of thanks would be enough for my collages Yiping, Julien, Cendrine, Soraya, Jigar and Tegawendé for helping and advising me during my stay at Bordeaux. Finally, I would like to thank my parents for their unconditional support and encouragement throughout the years, which made it all possible.

And to all my friends in Bordeaux and at the university sport center - *Merci Beaucoup!*

Abstract

The growth of the Internet as a large scale media provisioning platform has been a great success story of the 21st century. However, multimedia applications, with their specific traffic characteristics and novel service requirements, pose an interesting challenge in terms of discovery, mobility and management. Furthermore, the recent impetus to *Internet of things* has made it very necessary, to revitalize research in order to integrate heterogeneous information sources across networks. Towards this objective, the contributions in this thesis, try to find a balance between heterogeneity and interoperability, to discovery and integrate heterogeneous information sources in the context of Future Internet.

Discovering information sources across networks need a through understanding of how the information is structured and what specific methods they follow to communicate. This process has been regulated with the help of discovery protocols. However, protocols rely on different techniques and are designed taking the underlying network infrastructure into account. Thus, limiting the capability of some protocols to cross network boundary. To address this issue, the first contribution in this thesis tries to find a balanced solution to enable discovery protocols to interoperate with each other as well as provide the necessary means to cross network boundaries. Towards this objective, we propose ZigZag, a middleware to reuse and extend current discovery protocols, designed for local networks, to discover available services in the large. Our approach is based on protocol translation to enable service discovery irrespectively of their underlying discovery protocol. Although, our approach provides a step forward towards interoperability in the large. We needed to make sure that discovery messages do not create a bottleneck for the network.

In large scale consumer oriented network, service discovery messages could render the network unusable. To counter this, ZigZag uses the concept of aggregation during the discovery process. Using aggregation ZigZag is able to integrate several replies from different service sources supporting different discovery protocols. However, to customize the aggregation process to suit once needs, requires a through understanding of ZigZag fundamentals. To this end, we propose our second contribution, a flexible policy language that can help define policies in a clean and effective way. In addition, the policy language has some added advantages in terms of dynamic management. It provides features like delegation, runtime time policy management and logging. We tested our approach with the help of simulations, the results showed that ZigZag can both reduce the number of messages that flow through the network, and provide value sensitive

information to the requesting entity.

Although, ZigZag is designed to discover media services in the large. It can very well be used in other domains like home automation and smart spaces. While, the flexible pluggable modular design of the policy language enables it to be used in other applications like for instance, e-mail.

Keywords:- Service Discovery, Protocol Interoperability, Future Internet, Policy Language, ZigZag Middleware

Résumé de Thèse

La croissance de l'Internet en tant que plateforme d'approvisionnement à grande échelle d'approvisionnement de contenus multimédia a été une grande *success story* du 21e siécle. Toutefois, les applications multimédia, avec les charactéristiques spécifiques de leur trafic ainsi que les les exigences des nouveaux services, posent un défi intéressant en termes de découverte, de mobilité et de gestion. En outre, le récent élan de l'Internet des objets a rendu très nécessaire la revitalisation de la recherche pour intégrer des sources hétérogènes d'information à travers des réseaux divers. Dans cet objectif, les contributions de cette thèse essayent de trouver un équilibre entre l'hétérogénéité et l'interopérabilité, pour découvrir et intégrer les sources hétérogènes d'information dans le contexte de l'Internet du Futur.

La découverte de sources d'information sur différents réseaux requiert une compréhension approfondie de la façon dont l'information est structurée et quelles méthodes spécifiques sont utilisés pour communiquer. Ce processus a été régulé à l'aide de protocoles de découverte. Cependant, les protocoles s'appuient sur différentes techniques et sont conçues en prenant en compte l'infrastructure réseau sous-jacente, limitant ainsi leur capacité à franchir la limite d'un réseau donné. Pour résoudre ce problème, le première contribution dans cette thèse tente de trouver une solution équilibrée permettant aux protocoles de découverte d'interagir les uns avec les autres, tout en fournissant les moyens nécessaires pour franchir les frontières entre réseaux. Dans cet objectif, nous proposons ZigZag, un middleware pour réutiliser et étendre les protocoles de découverte courants, conçus pour des réseaux locaux, afin de découvrir des services disponibles dans le large. Notre approche est basée sur la conversion de protocole permettant la découverte de service indépendamment de leur protocole de découverte sous-jacent. Toutefois, dans les réaux de grande échelle orientée consommateur, la quantité des messages de découverte pourrait rendre le réseau inutilisable. Pour parer à cette éventualité, ZigZag utilise le concept d'agrégation au cours du processus de découverte. Grâce à l'agrégation, ZigZag est capable d'intégrer plusieurs réponses de différentes sources supportant différents protocoles de découverte. En outre, la personnalisation du processus d'agrégation afin de s'aligner sur ses besoins, requiert une compréhension approfondie des fondamentaux de ZigZag. À cette fin, nous proposons une seconde contribution: un langage flexible pour aider à définir les politiques d'une manière propre et efficace.

Le Chapitre 2 présente le contexte de notre travail. Pour faciliter la discussion, nous l'avons divisé en trois parties. La première partie donne un bref aperçu d'un service et décrit le processus de découverte de service. Nous avons décrit également les différents modes de recherche utilisé par les protocoles actuels de découverte de service. Nous présentons également la mise au point d'un certain nombre de protocoles de découverte de service populaires et les classifions soit en protocoles pour réseaux locaux (SLP, UPnP, Bonjour, WSDD) ou pour réseaux grande échelle (UDDI, SSDS, JXTA, INS/TWINE). En plus, nous passons en revue chacun d'eux mettant en évidence leurs caractéristiques standards, et identifiant leurs principales limitations. Durant cette phase de passage en revue, nous avons constaté qu'aucun des protocoles étudiés n'était interopérable.

La deuxième partie présente six modèles de contexte différents en fonction de leur structure de données, à savoir: Clé-valeur, à base de Balises, Graphique, Orienté-objet, basé sur la logique and basé sur l'Ontology. La simplicité de la structure de donnée clé-valeur facilite la gestion des informations de contexte. Malheureusement, son manque d'expressivité empêche toute déduction du contexte d'information considéré et sa structure plate ne prend pas en compte la définition de la relation entre les paramètres. En outre, l'absence de schéma de donnée et de méta-informations sur le contexte considéré rend ce type de modèle très difficile à réutiliser. Les Modèles à base de balise fournissent une validation partielle depuis qu'ils sont définis par des schémas. Cependant, comme les modèles clé-valeur, ils souffrent d'ambiguïté d'information, dans la mesure où ils ne supportent pas la définition de la relation entre les paramètres. Le principal inconvénient des modèles orientés objet, c'est l'absence de formalisme, puisque l'information est encapsulée. Par contre, ils offrent une distribution facile, permettant à chaque objet de se valider lui-même, et peuvent être intégrés directement dans des systèmes orientés objets existants, quoiqu'ils puissent être lourds en ressources. Les points forts des modèles graphiques sont leur efficacité dans la représentation de la structure des informations de contexte. En outre, ils sont aussi intuitifs et faciles à intégrer dans le modèle UML (Unified Modeling Language). Toutefois, ils présentent un faible niveau de formalisme, étant communément utilisés à des fins de structuration humaine. Pour les modèles basés sur la logique, le niveau de formalité est extrêmement élevé et les valeurs peuvent être distribuées, mais ils ne sont pas adaptés pour décrire l'incomplétude, l'ambiguïté ou la qualité de l'information. Pour atteindre cet objectif, le modeèle à base d'ontologie fournit une description de la conceptualisation explicite de la structure des données et la sémantique. Nous concluons que les modèles basés sur l'ontologie fournissent la meilleure solution pour la modélisation de contexte dans l'informatique ubiquitaire.

La troisiéme partie décrit ALICANTE, une architecture pour l'écosystème des médias du futur. L'architecture ALICANTE regroupe des acteurs clés dans plusieurs domaines appelés environnements (*utilisateur*, *réseau*, *service*). De plus, pour faciliter l'interaction à travers les domaines, deux nouvelles couches virtuelles sont considérées (Virtual HomeBox et Virtual CAN). Utilisant les caractéristiques combinées de l'environnement et des couches, il vise à fournir une propriété de sensibilité au Contenu à l'environnement réseau, de la sensibilité Utilisateur et Réseau à l'environnement service, et des contenu/prestations adaptées pour permettre une meilleure qualité d'expérience et de meilleures fourniture de services afin de bénéficier à tous les

acteurs impliqués. L'environnement de l'utilisateur présente des avantages pour l'utilisation du profil de l'utilisateur en permettant aux utilisateurs de basculer entre différents rôles. Cependant, il ne leur permet pas de découvrir et de publier des services utilisant des protocoles autres que ceux utilisés par le système Terminal. Néanmoins, les utilisateurs finaux devraient être en mesure de découvrir et de publier des services via le système Terminal quel que soit le protocole qu'ils utilisent. D'autre part, la capacité d'ALICANTE SE à séparer les données de la fonction de gestion donne un avantage unique permettant aux deux fonctionnalités d'évoluer séparément. Par ailleurs, avec l'aide des Service Registry (SR), ALICANTE fournit une infrastructure de services à grande échelle. Toutefois, cela ne signifie pas qu'il peut interagir et demander des services à partir de sources externes (ex: YouTube, Google, etc) ainsi que de plateformes existantes comme IMS (IP Multimedia Subsystem) et IPTV (Internet Protocol Television). Par conséquent, il existe un besoin pour un système de découverte interopérable sensible au contexte, qui peut intégrer de multiples approches hétérogènes de découverte dans des réseaux hétérogènes à grande échelle.

Le chapitre 3 traite de l'état de l'art actuel. Nous l'avons divisé en deux parties. La première partie décrit les différentes solutions qui répondent aux problématiques d'interopérabilité lors de la découverte de service. Nous examinons quatre solution interopérable à savoir; `ReMMoC`, `INDISS`, `z2z` et `Starlink`. `ReMMoC` est un intergiciel réflecteur reconfigurable et configurable dynamiquement. Il met en avant le concept de composants et interfaces proposées par `OpenORB`. Ces composants et interfaces permettent au middleware d'ajouter de nouvelles fonctionnalités à `ReMMoc`. En outre, il fournit également une interface de programmation d'applications générique (API) afin d'aider les développeurs à fournir des solutions interopérables. Cependant, `ReMMoC` demande aux développeurs de revoir toutes les applications existantes afin de les rendre conformes à l'API `ReMMoC`, ce qui est une tâche assez ardue. Cette contrainte particulière est surmonté avec `INDISS` qui est un middleware transparent qui assure l'interopérabilité avec les applications existantes sans les modifier. Les réponses `INDISS` sont des composants basés sur les événements. Toutefois, l'extension `INDISS` pour soutenir de nouveaux protocoles est une tâche difficile car il nécessite à la fois une connaissance approfondie des protocoles concernés, et aussi une compréhension importante de la programmation réseau de bas niveau. Bien que `ReMMoC` et `INDISS` pourraient être considérés comme un pas en avant dans le défi de la découverte de services interopérables, `z2z` et `Starlink` présentént de nombreuses fonctionalités pour permettre la traduction transparente d'un protocole à en un autre. `z2z` utilise une approche générative pour permettre la traduction de protocoles, tandis que Starlink s'appuie sur des automates k-couleur. En outre, ils offrent un système d'exécution optimisé et d'installations pour décrire les comportements de protocoles réseaux, les structures de message et les logiques de traduction. Ces fonctionalités viennent du fait qu'ils s'appuient sur un langage de définition de haut niveau qui cache le réseau de bas niveau et les détails et ne met en évidence que les propriétés clés nécessaires pour la traduction de protocole. Dans notre solution, nous nous appuyons sur la notion de composants introduit par `ReMMoC`, le réseau de surveillance basé sur des événements présentée par `INDISS` et fonctionalités fournis par `z2z` comme composant de traduction de protocole transparent pour offrir une solution de découverte interopérable pour l'internet du futur.

La deuxième partie traite des solutions basées sur l'ontologie. Nous avons examiné certains des modèles de contexte basé sur l'ontologie existants. Avec la maturité du Web sémantique, les modèles de contexte basé sur l'ontologie ont acquis beaucoup d'importance. Un contexte Ontology Language `CoOL` décrit des faits contextuels et des relations contextuelles en projetant la base conceptuelle de l'information Aspect-Scale Contexte modèle (`ASC`) pour les éléments de langage. Alors que `COBRA-ONT` est une collection des ontologies, OWL s'exprimé dans des systèmes sensibles au contexte. `COBRA-ONT` définit des concepts associés à quatre thèmes distincts mais liés: les lieux, les agents de l'emplacement et les agents des agents L'activité. D'autre part, `SOUPA`, développé par les mêmes auteurs que `COBRA-ONT`, traite de l'informatique omniprésente et se compose de deux parties: ❶ le noyau `SOUPA` qui détient les ontologies qui fournissent un vocabulaire commun pour différents environnements informatiques omniprésents et ❷ l'extension `SOUPA` qui contient des ontologies pour le vocabulaire spécifique à un domaine. Toutefois, afin de surmonter le problème de l'interopérabilité, `SOUPA` mappe ses concepts à certaines des ontologies externes bien connus tels que `FOAF` [BM07], `DAMLTime` [PH04], etc, le contexte ontologie `CONON` capte les caractéristiques générales des entités contextuelles de base comme (localisation, l'utilisateur , l'activité et entité de calcul) et utilise l'ontologie de domaine spécifique pour décrire les concepts liés à un domaine spécifique. Les modèles présentés ont montré que les ontologies peuvent être perçues comme des outils prometteurs vers une description adéquate de la représentation des données de contexte. Ainsi, dans notre solution, nous prévoyons d'utiliser et d'adapter le concept présenté par `CONON` qui sépare des informations de contexte à partir d'une générique spécifique à un domaine particulier. Cela nous permet d'améliorer notre solution pour fournir un service personnalisé de découverte en identifiant de façon unique les différentes entités et leurs propriétés.

Le chapitre 4 présente notre première contribution, le middleware ZigZag. Le middleware ZigZag est conçu pour fournir la découverte de services interopérables sensibles au contexte dans les réseaux de grande envergure. Pour atteindre cet objectif ZigZag est architecturé autour de 4 composantes principales (voir Figure 4.2), à savoir:

❶ SDP COMPOSANTE DU MONITEUR - Le moniteur SDP vérifie la disponibilité des différents points de desserte dans l'environnement local. Il s'appuie sur le fait que tous les SDPs utilisent une adresse de groupe de multidiffusion et un User Datagram Protocol (UDP) / Transmission Control Protocol (TCP) Port qui doit avoir été attribué par l'Internet Assigned Numbers Authority (IANA). Les deux ports affectés et les adresses de groupe multicast sont réservés et agissent donc comme une étiquette d'identification SDP permanente. En outre, les publicités de services sont mises en cache localement et sont associées à un Universally Unique IDentifier (UUID) qui doit être identifié de manière unique à travers différents nœuds de ZigZag.

❷ COMPOSANTE GESTION CONNECTEURS - Un connecteur traduit un Service Discovery Protocol (SDP) à un autre SDP. Il est spécifique à une paire de SDP. Ainsi, il existe autant de connecteurs que de paires de points de desserte entre lesquels l'interopérabilité est nécessaire. Un connecteur est un composant tiers. Actuellement, le composant de gestion Connecteurs s'appuie sur l'instanciation d'un ou plusieurs passerelles z2z [BRLM09] qui agissent comme des con-

necteurs. Cependant, ZigZag n'est pas étroitement liée à z2z, et peut compter sur tout autre traducteur. En outre, le composant de gestion Connecteurs recueille des statistiques sur les SDP utilisés pour prendre en charge le cycle de vie à grains fins de connecteurs instanciés. Il peut démarrer, arrêter, mettre en pause et reprendre des connecteurs selon les priorités de développement social le plus souvent détectés.

❸ LA RÉFÉRENCE COMPOSANT RÉSEAU - Le Link composant réseau offre une connectivité entre les nœuds en ZigZag. Il met en œuvre un protocole simple pour construire un arbre de distribution de données entre les nœuds ZigZag leur permettant d'échanger des messages multicast sur SDP, et des services dans chaque réseau local isolé. La complexité de la mise en œuvre des composants de liaison du réseau dépend des fonctions disponibles prises en charge par des couches de réseau inférieures. Actuellement, ZigZag soutient une superposition de données comme un exemple d'infrastructure de réseau pour l'internet du futur, qui fournit des primitives adéquates (rejoindre, quitter, mettre à jour, envoyer) pour créer et / ou maintenir un réseau logique entre les nœuds en ZigZag. En outre, ZigZag peut également être déployé sur différentes infrastructures de réseau, tels que des superposition P2P via la mise en œuvre des composants dédiés de liaison réseau.

❹ COMPOSANT AGRÉGATEUR - Le volet agrégateur rassemble une grande quantité de messages en provenance ou Ã destination de plusieurs connecteurs instanciés par la composante de gestion des connecteurs. Plus précisément, la composante agrégateur accumule toutes les réponses SDP provenant de différents nœuds en ZigZag à distance, et sélectionne celui qui correspond le mieux aux critères de la demande associée pour la transmettre ensuite au demandeur de service.

Ces composants sont connectés ensemble pour accomplir un processus de traduction réseau croisé qui est capable de traduire un SDP à l'autre selon les fournisseurs de services et la participation des consommateurs à travers des réseaux hétérogènes.

Le chapitre 5 présente notre deuxième contribution, le cadre des politiques. Comme la technologie évolue rapidement, il est devenu clair que les systèmes de gestion réseau existant et ne sont pas très bien adaptés pour faire face aux exigences automatisées et à l'auto-apprentissage dans un environnement axé sur les applications utilisateur. Il y a donc un besoin d'application / utilisateur de provisionnement conscients et de contrôle des ressources du système. Pour y remédier, nous avons conçu un cadre de politique pour permettre au middleware ZigZag à déléguer une tâche si elle ne peut satisfaire aux exigences de la nouvelle requête entrante. Comme le montre la Figure 5.3, le cadre politique se compose de trois éléments clés, à savoir, LE MOTEUR DE LA POLITIQUE, LE LANGAGE DE LA POLITIQUE ET LE GÉNÉRATEUR DE POLITIQUE.

❶ LE MOTEUR DE LA POLITIQUE régles se compose de quatre modules à savoir:
① `Analyseur Syntaxique Des Politiques` - Afin d'éviter lécrasement du système de vérification, la syntaxe d'exécution pourrait aller le long des voies avant que la politique

ne puisse être utilisé dans le système. À cette fin, l'analyseur de la politique adhère à la langue sans contexte décrit dans l'annexe II est fait en sorte que l'instance politique est validée avant qu'elle ne soit utilisé dans le système. En outre, l'analyseur de politique est aussi vérifié et valide les actions politiques de leurs paramètres avant leur stockage.

② `Politique Dévaluation` - La politique évaluateur a deux responsabilités principales. Tout d'abord, il est chargé de valider le résultat d'une requête entrante avec les politiques enregistrés. Et d'autre part, il est responsable du débit du système. Pour atteindre le premier objectif, la politique évaluateur utilise le concept appelé correspondance booléenne. Ceci correspondant à chaque condition de la politique qui est validée avec les mots-clés et les valeurs de la requête entrante, le résultat est stocké comme vrai ou faux.

③ `Policy Manager` - Gestion des systèmes distribués implique surveiller l'activité d'un système, la prise de décisions de gestion et de l'exécution des mesures de contrôle pour modifier le comportement du système. Toutefois, la taille et la complexité des grands systèmes distribués a abouti à une tendance à l'automatisation de nombreux aspects de gestion, dont les principales sont la capacité à s'auto-gérer la distribution de la politique. Ainsi, il ya un besoin croissant pour permettre aux gestionnaires de la politique de spécifier, de représenter et de manipuler l'information sur les politiques afin de permettre l'auto-gestion dans un environnement distribué dynamique. Dans ce sens, le directeur de la politique prend en charge les fonctions suivantes : *`ajouter, supprimer, activer, désactiver déplacer`*. Ces fonctions permettent au gestionnaire de la politique d'auto-gérer les entrées sur une télécommande ou un magasin de la politique locale.

④ `Action De La politique` - Les politiques encapsulent une représentation de l'information touchant au comportement des composants à l'aide de l'action politique. Ainsi, il est hautement avantages de préciser la portée d'une action politique à un domaine particulier. Cela permet au gestionnaire de politique d'identifier la politique qui s'applique à un domaine et ensuite utiliser cette information pour modifier le comportement des composants pour atteindre les objectifs du système. À cette fin, les actions politiques du magasins du module Domaines de renseignements sur les composants comme des objets d'action. En outre, ces actions objets sont ensuite utilisés pour consulter le gestionnaire de la politique avant l'exécution d'une action de la politique.

❷ LE LANGAGE DE LA POLITIQUE – Notre politique linguistique suit la recommandation de l'IETF *`Condition-Action`* paradigme. Les politiques ont la forme de: ❝ *S'il ya (un ensemble de conditions), alors (un ensemble d'actions) peuvent être effectuées* ❞. Par conséquent, notre langage de politiques repose sur deux concepts fondamentaux à savoir:
`policy_condition` et *`policy_action`*. En utilisant ces constructions, notre langage est capable d'influencer le comportement d'exécution d'un système. Le *`policy_condition`* est utilisé pour valider les différents articles de l'exécution, tandis que le *`policy_action`* est utilisé pour le débit du système.

❸ LE GÉNÉRATEUR DE POLITIQUE - Dans un environnement très répandu consistant en différent éléments, il est très difficile d'identifier de manière unique les différents éléments et leurs propriétés. Dans ce sens, la communauté sémantique a montré des résultats prometteurs avec

l'aide d'ontologies. L'ontologie fournit une description formelle et sémantique des informations de contexte en terme d'objets, concepts, propriétés et relations. L'utilisation d'ontologies a aidé Ã résoudre la question de l'identification unique des éléments et de leurs capacités. Toutefois, afin d'utiliser efficacement leur générateur de politique, on doit les présenter sous forme compréhensible par le système. Dans ce sens, et comme illustré à la Figure 5.5, le générateur utiliser le service d'un modèle. Les modèles fournissent des blocs de construction de politique obligatoires en laissant suffisamment de lacunes à combler en utilisant les informations dynamiques extraites de données modélisées basées sur l'ontologie.

Le chapitre 6, Dans cette thèse, nous avons proposé une nouvelle approche pour découvrir les services . Notre approche est basée sur la conversion de protocole pour permettre la découverte de services indépendamment de leurs protocoles de découverte de services sous-jacents. Nous introduisons ZigZag, un middleware de réutiliser et d'étendre les protocoles actuels de services de découverte, conçus pour les réseaux locaux, pour découvrir les services disponibles à travers les frontières du réseau tel que requis dans l'Internet du futur. Le middleware ZigZag peut être déployé en tant que solution autonome ou peut être intégré dans un contexte existant grâce à sa conception modulaire. En outre, le middleware peut être configuré pour découvrir les services basés sur diverses applications exigentes avec l'aide des politiques. Les politiques permettent aux développeurs de définir les conditions du système. Nous avons testé notre approche à l'aide de simulations et les résultats ont montré que ZigZag peut à la fois réduire le nombre de messages qui circulent à travers le réseau, et fournir des informations sensibles de la valeur à l'entité requérante.

Le Middleware ZigZag a été conçu à l'origine pour des services de multimédias de découverte dans un environnement hétérogène multiprotocole. Cependant, les avantages de ZigZag vont au-delà l'accès aux médias et peuvent être utilisés efficacement pour automatiser et permettre de créer des espaces intelligents. En outre, l'amélioration récente dans la technologie et la popularité de l'Internet des choses ont créé les conditions propices à l'éveil des structures intelligentes et les espaces intelligents. Smart Home de services comme le contrôle centralisé de l'éclairage, de HVAC (chauffage, ventilation et climatisation) des appareils; serrures de sécurité des portes et des portes et autres systèmes. Cependant, la plupart des fournisseurs de ces services reposent sur des protocoles différents. Dans ce sens, ZigZag peut aider les vendeurs à coordonner l'interaction entre les différents services. En outre, ZigZag peut également permettre aux utilisateurs de contrôler divers services avec leurs smartphones et tablettes, de fournir une meilleure commodité et le confort de la paume de leurs mains. Bien que le langage de la politique a été conçu pour aider les développeurs à écrire des politiques pour ZigZag middleware, sa conception flexible lui permet d'être utilisé dans d'autres applications. Ainsi, dans une application e-mail, les développeurs peuvent charger un plugin, analyser un e-mail qui permettrait aux politiques de trier ou filtrer les messages e-mail dans une boîte de réception en fonction des préférences des utilisateurs.

Mots clés:- découvrir les service, protocole d'interopérabilité, Internet du futur, Langage de la politique, Middleware ZigZag

Contents

List of Figures

List of Tables

List of Tables

Acronyms

API Application Programming Interface 32
CF Component Framework 32
CAN Content-Aware Network 27
CIM Common Information Model 66
CDN Content Delivery Networks 26
CBIM Common Baseline Information Model 18
DMTF Distributed Management Task Force 66
DSL Domain Specific Language 37
DHCP Dynamic Host Configuration Protocol 11
DNS Domain Name System 11
DAs Directory Agents 10
DFA Deterministic Finite Automata 37
EU End-User 24
FI Future Internet 1
HB Home-Box 26
HD High Definition 2
HTTP HyperText Transfer Protocol 10
IETF Internet Engineering Task Force 65
FI Future Internet 1
IMS IP Multimedia Subsystem 28
IP Internet Protocol 11
IPTV Internet Protocol Television 28
IDL Interface Definition Language 34
INS Intentional Naming System 15
INR Intentional Name Resolver 15

SMTP Simple Mail Transfer Protocol 13
TCP Transmission Control Protocol 4
UDP User Datagram Protocol 4
UE User Environment 24
UPnP Universal Plug and Play 10
URL Uniform Resource Locator 13
UDDI Universal Description Discovery and Integration 13
URL Uniform Resource Locator 13
UML Unified Modeling Language 18
UDC User Data Convergence 18
UUID Universally Unique IDentifier 4
VCAN Virtual CAN 27
VoD Video On Demand 1
WSDL Web Service Description Language 34
WSDD Web Service Dynamic Discovery 11
XML eXtensible Markup Language 10

Chapter 1

Introduction

"Everything is a file"

–Linux Philosophy

Contents

Since its design over 40 years ago, the Internet has been used as the medium and infrastructure for information exchange by billions of users as well as by hundreds, even thousands of different applications and services. Diverse applications like e-mails, e-commerce, secured remote access, Video On Demand (VoD) and interactive video outline a very successful communication approach in the large. However, the Internet has its fair share of limitations. Some of these limitations include larger-scale service provisioning, management and deployment; facilities for device and service mobility; support for better Quality of Service (QoS) and use of Service Level Agreement (SLA) for improving service management and enhance security features. These limitations are not new as the Internet was traditionally designed to interlink and access documents over connected networks. Thus, there is a growing need to re-design at least partially the current Internet's architecture [AAB+11]. This, in turn, will enable future networks and services as well as novel technologies to cope with new demands. Currently, several research frameworks all over the world have started to work towards this objective, leading to a lot of ongoing research to re-design the current Internet architecture [NSF10, FIP94, AKA06]. They are all working towards a common goal, build the Future Internet (FI) architecture.

Future Internet is presently seen as a large scale infrastructure that can provide dependable management, instantiation and interoperation among heterogeneous networks. These networks support numerous users, devices and services and may depend on different communicating technologies. Thus, to empower interactions among different entities across networks the Future

Internet should be information driven and rely on services. Furthermore, service consumers should be able to use *anytime*, *anywhere*, remote services on *any device* regardless of their underlying technology . Service Discovery as it turns out is a crucial initial step towards this objective. However, the vast bulk of services primarily connected to the Internet have not been designed to interact seamlessly with each other as they may rely on different Service Discovery Protocols (SDPs). Therefore, to realize the future vision of *anytime, anywhere, any device*, the discovery process should address the issue of protocol heterogeneity to enable future services.

Additionally, end users demand a better Quality of Experience (QoE) as well as personalized services to cater to their unique needs [HK02]. This requires prior knowledge about users' preferences, device capabilities and network characteristics. Even though contextualizing the end users, i.e. creating and aggregating specific information characterizing their context, has been very successful, current SDP's have not been able to exploit such advantages. Therefore, there is a need to combine the capabilities of service discovery with the advantages of context information to enable highly sophisticated services customized to end-user needs. Towards this, we believe that *policies* can be used to combine the capabilities of service discovery and context information to enable customized services for the future. Furthermore, they can also be used to provide dynamic and flexible management functionality that can deal with the increasing size and complexity of proposed Future Internet architectures.

1.1 Motivation

To highlight our motivation, we present a computing scenario to emphasize on various complexities that exist in large scale heterogeneous networks. The proposed example (see Figure 1.1) has different heterogeneous networks connected to an overlay forming a large scale distributed infrastructure. Each network uses the assistance of a gateway to connect to the overlay. As illustrated, each network support different Service Discovery Protocol (SDP) on diverse devices. Instances of services are advertised using contrasting SDPs making it difficult for clients to discover available services. This situation is aggravated, as these networks may host thousands of users, services and devices simultaneously. Furthermore, a device is a heterogeneous entity, having different display size, processing power, memory and network access capability. This requires services to also adapt to different devices capabilities.

Moreover, user generated multimedia content and services using personal mobile devices are gaining a lot of interest, indicating an increase in production and consumption of personal multimedia content [BDS08] . In fact, End-Users may have different requirements and service preferences. For instance, a user may want to view a High Definition (HD) video on a device capable of playing only Standard Definition (SD) content; as End-User have the capability to use diverse devices (mobile phone, tablet, laptop) to access available services. Indeed, devices have different display capabilities and may support different audio/video codec other then the ones needed to consume a particular service. This raises the need to adapt the content to meet users' current needs based on his/her current context or situation. Additionally, as multimedia

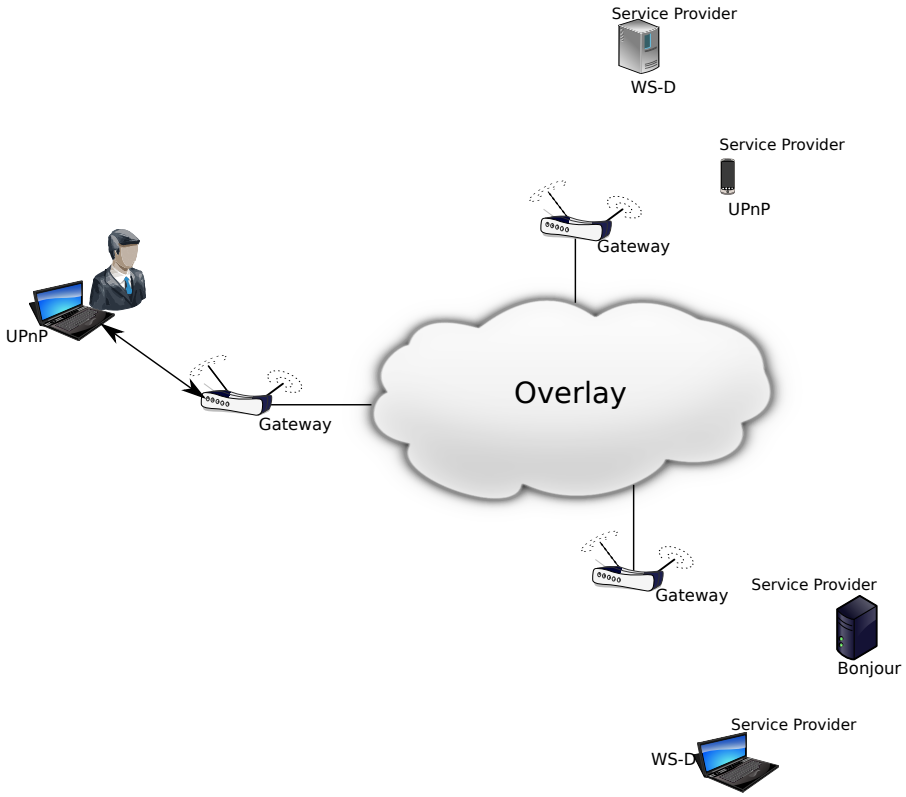

Figure 1.1: Motivation: Service Discovery in Heterogeneous Networks

applications are resource consuming, device processing capability (CPU, memory) and network conditions (bandwidth, bit rate, packet loss) may change at runtime due to varying application requirements.

Given, the aforementioned complexities, it is difficult to anticipate and manage the runtime behavior of multimedia services. Such runtime administration requires an integrated approach to manage protocol interoperability, context gathering and utilization as well as content adaptation. The work presented in this thesis takes this into account and proposes an alternate service discovery approach that can discover services in large scale heterogeneous networks with the help of ZigZag middleware [RBRN12]. The ZigZag middleware is designed in a modular way and can be easily integrated with existing solutions. It aims to manage the heterogeneity of various SDPs by integrating any existing interoperability system to translate, on the fly, one SDP to another in the context of Future Internet. Furthermore, we also propose a policy language [RCB+12] that enables us to manage different components of ZigZag at runtime. This, in turn, manages various resources and enables discovery and consumption of highly scalable multimedia services. Further, the policy language leverages on ontologies to enable context-aware service discovery in a highly dynamic and distributed environment like the Future Internet.

1.2 Contribution

This thesis proposes two contributions. Firstly, we propose a solution to seamless discovery services in large scale networks with the help of a middleware. Furthermore, we show the applicability and flexibility of our approach in large scale service infrastructure. Secondly, we propose a policy language that enhances the middleware solution to provide service personalization and runtime management.

Seamless service discovery
Today, a large number of services and content is available on the Web. A key challenge to seamlessly access these services is to discover them regardless of the underlying technology or devices. Service Discovery protocols enable devices to discover services without any prior knowledge. However, devices rely on different SDPs making it difficult to discover all available services. Moreover, some protocols are designed for local networks, hence are not able to cross network boundaries. Therefore, there is a need to address protocol heterogeneity to enable seamless discovery of services. Furthermore, existing discovery approaches are designed with a particular target infrastructure size in mind, regarding network size and number of consumers and services. Our consideration of scale pertains to *large size* networks and global service infrastructures with thousands of users. Towards this, we rely on an infrastructure defined by a large-scale European project ALICANTE [ALI13].

ALICANTE proposes key elements that provide support for large-scale service infrastructure. These elements provide the criteria to examine the target infrastructure size and allow us to evaluate the ability of our discovery approach to cope with large scale networks. In such large-scale service infrastructure the communication overhead of discovery can reduce the effective use of network resources to an unacceptable level. Consequently, the discovery system should be able to cope with a large amount of consumer requests in a consumer friendly service infrastructure. Thus, we need to consider different strategies that optimize the communication overhead and provide an interoperable solution for large-scale networks like the Future Internet. To this end, we propose the ZigZag middleware. The ZigZag middleware solution, instantiate any existing interoperability system to translate, on the fly, one SDP to another. Additionally, it also aggregates service information with the help of polices, thereby reducing number of messages on the wire. Chapter 4 highlights in details the different components and advantages of ZigZag middleware.

Context-aware personalization
Over the last few years, the improvements in technologies, has led to a massive increase in user generated content and services. These services need to take full advantage of device characteristics (display, CPU, memory) to provide users with best QoE. Furthermore, services should be aware of users current situation and take into account his/her preferences. Towards this, ontologies provide a promising solution by supporting formal, explicit, machine-processable semantic definition with support for further knowledge discovery. The Knowledge discovery enables services to be personalized. Further, it brings benefits for users by matching his stated and learned

preferences. However, some of the existing SDPs do not support context-awareness. In order to enable these protocols to utilize context information, we propose a policy language that can combine the advantages of context information with service discovery to provide highly customized services. Chapter 5 presents in details the different components of the policy framework, the language syntax and its usage.

1.3 Thesis Structure

To efficiently introduce our work, the identified issues and our contribution to address them, this document is structured as follows:

Chapter 2 introduces the background to our work. To facilitate the discussion we have separated it into three parts. The first part gives a brief overview of a service and describes the process of service discovery. It also describes different discovery modes used by current service discovery protocols. Further, it shows the classification of SDPs and highlight existing issues. The second part presents six different context models based on their data structure. It concludes by asserting how ontology based models provide the best solution for context modeling in ubiquitous computing. The third part describes ALICANTE, an architecture for Future Media ecosystem. It presents key entities proposed by ALICANTE and highlights their advantages and identifies some limitations. Finally, it concludes by describing the need for interoperable context-aware service discovery in large scale heterogeneous networks.

Chapter 3 discusses the current state of the art. In order to simplify the understanding for the reader, we have divided it into two parts. The first part describes different solutions that address protocol interoperability during service discovery. Furthermore, it compares the features of each of the solutions. And finally, concludes by describing how generative based solution can be used to provide transparent interoperability during service discovery. The second part discusses ontology based solutions and concludes by describing how the concept brought forward by CONON can be utilized in our solution.

Chapter 4 presents our first contribution, the ZigZag middleware. To highlight the importance of our work, it discusses the need for ZigZag middleware approach for service discovery in large scale networks. It then describes the ZigZag middleware architecture and highlights the relevance of aggregation for service discovery in large scale networks. Furthermore, it presents the evaluation of ZigZag approach with the help of simulations. And finally, it describes the integration of ZigZag middleware in ALICANTE.

Chapter 5 presents our second contribution, the Policy framework. To show the importance of the policy framework, it describes the need for policy in ZigZag middleware. It then presents the policy framework architecture. Furthermore, it introduces our policy language and highlights its advantages. It also describes how we use the policy framework to generate context-aware policy

conditions. And finally, it describes how we can use the policy language to share and utilize resources in large scale networks.

Chapter 6 concludes this thesis and offers a perspective for future work.

Chapter 2

Background

"Mistakes are the portals of discovery"

–James Joyce

Contents

2.1 Brief history

A *service* is defined as a hardware/software resource that can perform one of more functions for an user/application or other devices over a network. *Service Discovery* is the process of finding the location of an entity in the network that provides access to its available services. The process

of *Service Discovery* is accomplished with the help of two basic entities namely; ❶ the *Service Consumer* and ❷ the *Service Provider*. *Service Consumer* represents the entity that is interested in finding and using the service (also called *Client* or *User agent*). While on the other hand, *Service Provider* represents the entity that hosts and offers various services (also called *Server* or *Service agent*). However, to cater to a larger audience, a *Service Provider* sometimes rely on a third entity known as *Service Registry* (also called *Directory* or *Directory agent*). Figure 2.1 illustrates the interaction among the three entities. Furthermore, *Service Discovery* also aims

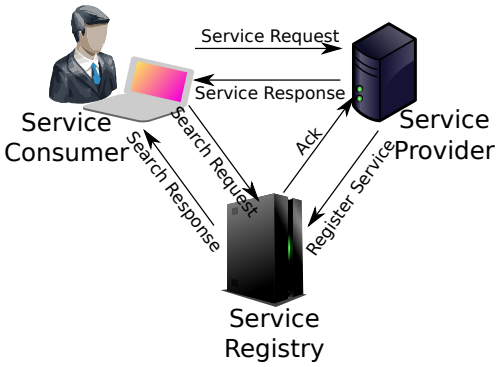

Figure 2.1: Service Discovery Interaction [CHRT04]

to make the devices discovering the services intelligent so that they may find available services without any previous knowledge of either their location or their characteristics. To this end, Service Discovery supports two modes of operation, namely; *active* and *passive*.

Active

In Active service discovery mode, devices find services by sending a request on the network and monitoring its response. If the address of the services is known in advance then *unicast* communication is used else the request is sent by *multicast* [Dee88] or *broadcast* [Mog84].

Passive

In Passive service discovery mode, devices find services by listening for service advertisement on a dedicated port on the network. The passive mode uses either *multicast* or *broadcast*.

These discovery modes are used by almost all service discovery protocols. However, some protocols support discovery process in both modes, while others rely on only one of them. These modes are configured to optimize application response time for different networks. As mentioned before, SDPs rely on *Service Registry* to cater to a larger audience. Furthermore, it is also used to organize and store service information. This enables other entities (*service consumer* or *Service Provider*) to use it to search or to publish their service information. Depending on the required network configuration, the *Service Registry* can be used in one of the follows modes:

Centralized
In the centralized approach, a single entity contains all service registry entries. This model follows a traditional client/server approach where the registry acts as a server. The advantage of having one global registry is that, the management of distributed resources is done centrally. Furthermore, the control of authority and quality is easily managed, and service description is consistence in style and presentation. Moreover, authorization and security issues are handled quite easily as they are administered from a central point. The disadvantages however is that, this approach does not scale very well as there is a limit for the number of resources which are managed and registered in the global registry.

Decentralized
In the decentralized approach, there are only local registries, one for each administrative domain. The advantage of this approach is that the management of a central registry is not required anymore. That implies that centralized administration is not necessary as it is done locally by each registry. Issues regarding security, complexity and authorization are managed for each domain independently.

Hybrid
The hybrid approach has the benefit of shared management and administration of local and global registries. The local registries are responsible for the management of their own resources but the global registry is responsible for high-level management. Furthermore, when a service is absent in the local registry within the administration domain, a request is sent to the global registry inquiring where to find this particular service. It (local registry) then can directly contact the responsible registry. Issues such as authorization, security and complexity are easily handled. With respect to scalability, this approach scales well up to the limit of storage capacity of the global registry.

The next section reviews some well known service discovery protocol implementations and highlights their standard features.

2.2 Popular Service Discovery Implementations

Service Discovery Protocols are usually designed taking the underlying network topology into account. To optimize service discovery for different networks they are broadly classified as SDPs for either *Local Area networks* or *Large Scale Networks*.

2.2.1 Local Area Networks

A Local Area Network (LAN) is an interconnection of devices in a limited area such as a home, school, computer laboratory [KPS02]. In a LAN, devices discover services without any prior knowledge about their location or characteristics using the following SDPs:

Service Location Protocol (SLP)

The Service Location Protocol (SLP, srvloc) [GPVD99] is a service discovery protocol that allows computers and other devices to find services in a local area network without prior configuration. SLP has been designed to scale from small, unmanaged networks to large enterprise networks. It has been defined as a Standards Track document [Gut02]. SLP is a binary packet-oriented protocol. Most packets are transmitted using UDP [Pos80], but TCP [APS99] can also be used for the transmission of longer and reliable packets. Because of the potential unreliability of UDP, SLP repeats all multicasts several times in increasing intervals until an answer has been received. All devices are required to listen on port 427 for UDP packets, Service Agents (SAs) and Directory Agents (DAs) should also listen for TCP on the same port. SLP can be configured for both Active and Passive Service Discovery modes. Some other standard features supported by SLP are highlighted in Table 2.1.

Service Location Protocols (SLP)	
Type of Network	–Local Network
Architecture	–Centralized
Storage of Service information	–On DA –On SA
Search Methods	–Both active and passive discovery
Event notification	—
Service description	–Uses Service templates registered with IANA
Service selection and usage	—
Fault tolerance and mobility	–Service Registration Lifetime must be refreshed or they expire.
Network Scalability	–More DA –scope mechanism for service grouping
Security	–Optional authentication of DA and SA (using digital signatures)

Table 2.1: SLP : Standard features

Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) initially promoted by Microsoft, is currently maintained by the UPnP Forum founded in 1999. The first version of the UPnP device architecture was released in 2000 [PFK+08]. The architecture is made up of six functions namely; *Addressing*, *Discovery*, *Description*, *Control*, *Eventing* and *Presentation*. Using these functions UPnP provides a decentralized, open networking architecture that uses TCP/IP and Web technologies (HyperText Transfer Protocol (HTTP) [FGM+99] ,Simple Object Access Protocol (SOAP) [BEK+00] / eXtensible Markup Language (XML)) to enable seamless networking in managed and unmanaged small networks. However, to discover and announce the presence of services the UPnP

Discovery function relies on Simple Service Discovery protocol (SSDP). Using TCP port 2869 or UDP port 1900, SSDP allows devices to advertise its services to *Control Points*. *Control Points* part of the *Control* function, acts on the service consumer's behalf, catch the interesting service announcements and can also initiate queries based on service consumer's needs. Table 2.2 highlights the standard features.

Universal Plug and Play (UPnP)	
Type of Network	–Enterprise Network
Architecture	–Decentralized
Storage of Service information	–On every control point
Search Methods	–Both Active and Passive discovery for finding services.
Event notification	–Eventing Mechanism
Service description	–XML description, based on UPnP template language.
Service selection and usage	–Usage message encapsulated in SOAP.
Fault tolerance and mobility	–Expiry time for advertisement *Device unavailable* notification.
Network Scalability	—
Security	—

Table 2.2: UPnP : Standard features

Bonjour

Bonjour (formerly Rendezvous) is a device and service discovery protocol developed by Apple computers. It relies on multicasting to provide service and device discovery among computers, electronic appliances, and other networked devices (e.g., printers, fax machines, etc.). Bonjour uses the Internet Protocol (IP) and also has the capability of automatically assigning IP addresses to networked devices, even without the help of a Dynamic Host Configuration Protocol (DHCP) server. Bonjour core is entirely based on the multicast Domain Name System (DNS) service discovery – mDNS-SD [CK13]. Bonjour [Che11] in an ad hoc network can resolve service names, in addition to host names, to IP addresses without relying on DNS servers [Dro97]. In mDNS-SD, clients multicast their DNS-like queries specifying the service type they are looking for, the domain where the service resides, and the preferred communication protocol. Service providers respond to those queries through DNS service records [GVE00] [Moc87]. See Table 2.3 for some other standard features.

Web Service Dynamic Discovery (WSDD)

WSDD [MK09] is a service discovery protocol co-developed by BEA Systems, Canon, Intel,

Bonjour	
Type of Network	–Enterprise Network.
Architecture	–Centralized.
Storage of Service information	–Service Producer.
Search Methods	–Both active and passive discovery.
Event notification	—
Service description	–name/value pairs.
Service selection and usage	–mDNS query.
Fault tolerance and mobility	–update mDNS records.
Network Scalability	—
Security	—

Table 2.3: Bonjour : Standard features

Microsoft, and webMethods, Inc. It defines a multicast request which is used to discover a set of endpoints that match the request. A request is basically a XML message called *probe* which contains the search criteria to find a service. WSDD uses the well known network multicast port 3702. However, it has an advantage over other protocols, such as SLP, i.e., it can dynamically switch from active to passive mode and vise versa. Table 2.4 highlights the other standard features supported by WSDD.

WS-Dynamic Discovery (WS-DD)	
Type of Network	–Enterprise Network
Architecture	–Centralized
Storage of Service information	–On Target Service or Discovery Proxy
Search Methods	–Both Active and Passive discovery for finding services.
Event notification	– Using other WS-* standards
Service description	–XML description.
Service selection and usage	–Usage message encapsulated in SOAP.
Fault tolerance and mobility	–Hello messages are used as update.
Network Scalability	—
Security	– Using other WS-* standards

Table 2.4: WSDD : Standard features

2.2.2 Large Scale Networks

A Large Scale Network is an interconnection of networks over a large geographical area, such as the Internet. The remainder of the section describes some SDPs used to discover services in

such networks.

Universal Description Discovery and Integration (UDDI)
UDDI [CHRT04] is a cross-industry initiative that aims at creating a global, platform-independent, open registry standard for distributed Web-based publication and discovery of Web services. A UDDI registry allows a provider to register information about itself and the services it provides. Business information might include a company name, contact information, and description. Besides a classical keyword-based search, a UDDI registry can be browsed according to three different modalities, namely through *white pages* (Company contact information, such as name, description of the business etc.), *yellow pages* (businesses categorized by standard taxonomies) and *green pages* (the document with technical information about exposed services as well as pointers to various files and Uniform Resource Locator (URL) [BLMM+94] - based discovery mechanisms). UDDI registries allow any protocol to be associated with a service. The most common Web service protocols (i.e., HTTP, SOAP, and Simple Mail Transfer Protocol (SMTP) [Kle08]) are pre-registered in UDDI registries as *tModels*. A *tModel* describes a *technical model* representing a reusable concept, such as an abstract service type, a protocol used for services to communicate, a taxonomy system, etc. See Table 2.5 for some other standard features.

Universal Description Discovery and Integration (UDDI)	
Type of Network	–large scale network.
Architecture	–Centralized.
Storage of Service information	–Service Producer.
Search Methods	–Both active and passive discovery.
Event notification	—
Service description	–XML description.
Service selection and usage	–Usage message encapsulated in SOAP.
Fault tolerance and mobility	–Lifetime for service registration.
Network Scalability	–Multiple shared hierarchies.
Security	–UDDI security API (SSL).

Table 2.5: UDDI : Standard features

Secure Service Discovery Service (SSDS)
SSDS [CZH+99] was designed and developed by the Computer Science Division, University of California, Berkeley, in 1999. The architecture also includes *SDS Service*, *Certificate authority* and *Capability Manager* apart from the Service Consumer and Service Provider. *SDS Servers* are DAs, and form a tree hierarchical structure. Each *SDS Server* accumulates service descriptions from services or other *SDS servers* (also called Child SDS servers) using Bloom filters [Blo70]. The Bloom Filters basically aggregates the service Description using bit-wise OR and propagates the aggregated Bloom filters to its parent *SDS servers*. *Certificate Authority* is a trusted central component in the architecture which is responsible for verifying the digital signatures used to

establish the identities of different components in the SDS architecture. *Capability Manager* uses access rights (capabilities) that control the visibility of services to the service Consumer. Capabilities are signed messages, indicating that a particular class of service descriptions can be discovered by the specific service consumer. Table 2.6 highlights some other standard features.

Secure Service Discovery Service (SSDS)	
Type of Network	–Wide Area
Architecture	–Hierarchical structure of SDS servers.
Storage of Service information	–SDS Servers
Search Methods	–Passive Discovery of SDS Servers –Client queries are routed through the hierarchy
Event notification	—
Service description	–XML aggregation of service description using Bloom–filtered crossed terminals (BCT)
Service selection and usage	—
Fault tolerance and mobility	–Soft state of service announcements.
Network Scalability	–Multiple shared hierarchies. –Shedding server load by spawning on a nearby node.
Security	–Authentication of endpoint via digital signature, privacy by encryption and access rights via capabilities.

Table 2.6: SSDS : Standard features

JXTA

JXTA is a P2P platform based on a set of services and protocols to develop P2P applications. JXTA was originally created by Sun Microsystems [TAA+03] in 2001 and since has been an open-source project. Conceptually, JXTA provides a distributed model for peers to discover each others and interact, and an infrastructure to establish and manage routes and communications between peers across heterogeneous large-scale networks. Peers on top of the JXTA platform operate on an overlay network which provides network transparency. JXTA has several standard components: *advertisements*, *peers*, *peer groups*, *pipes*, six basic protocols and strong security features. JXTA relies on XML [BPSM+97] for description of its resources, protocols and components, called *advertisements*. Therefore, any system that can parse XML can potentially use JXTA. *Peers* in JXTA are divided in two classes: *edge peers*, which are normal user peers, often characterized by unstable behavior or lack of resources, and *super peers*. *Super peers* are further divided into *relay peers* that enable peers interaction over firewalls and Network Address Translation (NAT) boxes as well as *rendezvous peers*. The latter acts as coordinator and stores *advertisements* of the dependent *edge peers*. *Rendezvous peers* are almost classical *super peers*

and create a further overlay on the basic JXTA network. Additionally, the system provides virtual communication channels, called *pipes*, to send messages and data. *Pipes* are asynchronous, unreliable and unidirectional. The *pipes* offer two modes of communication: point-to-point and propagation. The propagation pipe defines a connection that has one output and several inputs. Table 2.7 highlights the other standard features.

JXTA	
Type of Network	–Wide–area (P2P)
Architecture	–Virtual Network Overlay
Storage of Service information	–Rendezvous peers
Search Methods	–Loosely consistent DHT combined with a limited range rendezvous walker
Event notification	—
Service description	–XML
Service selection and usage	—
Fault tolerance and mobility	–index replication among resolvers. –periodic updates and heartbeat among rendezvous.
Network Scalability	–Scalable distributed hash indexing. –No frequent updates as DHT is loosely consistent.
Security	–Certificate Authority based trust model. –Client-server authentication via digital signatures.

Table 2.7: JXTA : Standard features

Intentional Naming System (INS)/Twine

INS/Twine [BBK02] was developed by the MIT Laboratory for Computer Science in 2002 as an enhancement of INS [AWSBL99]. INS/Twine acts similarly to INS. However, it uses the Chord [SMK+01] indexing scheme at the Intentional Name Resolver (INR) overlay for better performance and scalability. INRs form a chorded ring that follows the Chord specification. The INR network address is known to all clients and service providers. A service provider contacts an INR to register its name-specifier and service information. This INR splits the name-specifier into strands (path from root to leaves), generates a 128-bit numeric key for each strand using the Message Digest (MD5) [Riv92] hash algorithm, and uses Chord to distribute these keys in appropriate INRs along with the information to which the key refers. For a better tolerance for INR failures, a number of replicas of this information are distributed among INRs. Table 2.8 highlights standards features supported by INS/Twine.

Intentional Naming System (INS)/Twine	
Type of Network	–Large and dynamic environments.
Architecture	–Overlay network of resolvers which form a DHT.
Storage of Service information	–Each resolver holds a range of keys and their values.
Search Methods	–Service discovery messages are routed in O(log N) hops.
Event notification	—
Service description	–Hierarchies of attribute value pairs.
Service selection and usage	—
Fault tolerance and mobility	–Each strand is stored on multiple nodes. –Hybrid state management scheme.
Network Scalability	–Hash-bashed partitioning of resource descriptions among resolvers.
Security	—

Table 2.8: INS/Twine : Standard features

2.2.3 Critical Analysis

In the above section, we reviewed some of the well known service discovery protocol implementations by both academia and industry. Some of these protocols have been very successful in local area networks while others have shown great promise for discovery in the large. However, during the review process a common pattern started to emerge. We noticed that, all the above protocols lack support for *interoperability*. As mentioned previously, we think that FI is a large scale heterogeneous network supporting different protocols simultaneously. Thus, making these protocols interoperate with each other will be highly advantageous for future services. It is worth mentioning that, even though some protocols use XML to describe their service information, they rely on their different schema and proprietary techniques, making them difficult to interoperate with each other as they are.

2.3 Context Models

Context is a generic term that, until now, has not been given a clear standard definition. Nevertheless, in the field of ubiquitous computing, several definitions have been provided in the literature [CK+00, Pas98, SAW94, Sch95, SBG99]. However, a very well adopted definition proposed by *Dey* [DAS01] defines *context* as any information that can be used to characterize the situation of an *entity*. An *entity* can be a *person*, *place*, or an *object* that is considered relevant to the interaction between a user and an application, including users and applications themselves. Once context information is identified, an important task is to define how information will be useful for applications. To this end, a generic definition adopted by research community states that

a system is *context-aware* if it uses context to produce information and / or services relevant to the user. The relevance depends on the tasks requested by the user. However, to be processed by computational entity, context information should be formally described in a contextual model. The authors [SLP04], classified context models based on the data structure used to represent and to exchange context between the system entities. They identified six context modeling approaches, namely; key-value models, Markup scheme models, graphical models, object-oriented models, logic-based models and ontology-based models. These models are briefly explained and illustrated through examples in the following section.

Key-value model
One of the simplest and well used context model is the key-value model. In the key-value model (also known as flat model) context information is represented as key-value pairs (name of the context information and its actual value). The simplicity of this model has led to its adoption in several systems. Active Badge system [SAW94] used it to exchange context information formatted as environment variables, while Context Toolkit [DAS01] uses it to manage context widgets. For instance, in Context Toolkit, each widget represents a state composed of a set of parameters characterizing the context information. Furthermore, it is also used to represent the associated behavior so that it can notify the applications of variation in its parameters. For example, in Context Toolkit, the state of `IdentityPresence` widget models the presence information using three key value pairs: the *managed location*, the identifier of the last *detected user*, and the *time* when the detection occurred. To highlight the associated behavior feature, the widget notifies the application about the arrival or departure of an user through the aforementioned key-value pairs. The IETF Media Feature Set standard [Kly99] also uses the key-value model to describe the terminal characteristics and user preferences. However, the context information is described and represented as boolean expressions.

Markup scheme model
The models are characterized by a hierarchical data structure. The context information is organized into elements identified by their tags, which are associated with attributes and contents. Recursively, an element can itself contain other elements. However, to include elementary constraints and relationships among elements W3C introduced Resource Description Framework (RDF) [LS+99]. The markup scheme model is often used for defining profile information (User, device). One of the well known markup scheme model is the GUP [ETS11b, ETS11c, ETS11d] defined by the 3GPP which uses the W3C XML Schema [Tho04, BMC+04] for harmonizing the usage of user-related information coming from different entities. The standard does not impose any classification of the information to be included in the profile. The Moving Picture Experts Group (MPEG) defines two standards based on markup scheme model to ensure multimedia interoperability namely; the MPEG-7 [MKP02] and the MPEG-21 [BDD05]. The MPEG-7 standard provides tools for describing the multimedia resources. Its description includes various information on the multimedia content such as its classification, creation (title, creators, etc..), usage (history of use, copyright, etc..), storage (format, encoding, etc..), as well as structural aspects (spatial components, temporal or spatio-temporal content), conceptual as-

pects (objects, events, etc..) or some low-level characteristics (colors, textures, etc.). Thanks to these descriptions, MPEG-7 allows multimedia content to index resources strongly based on the content. Furthermore, it improves the search and discovery functions. On the other hand, the MPEG-21 standard allow users to access, consume, share and manipulate multimedia content in an efficient, transparent and interoperable way. To be precise, the standard defines an open architecture that covers the entire distribution and consumption chain of multimedia content.

Graphical model

Unified Modeling Language (UML) is one of the most used generic graphical modeling tool. It can very well be used to model context information graphically. The context entities and their processing are represented as UML diagrams (class diagram, use case diagram, sequence diagram, etc.). A common example used to highlight the power of UML is the graphical modeling of air traffic control [BKE03]. However, to showcase the importance of graphical models to model context information the authors [HIR03] extended the Object Role Modelling (ORM) [HMM08] to allow facts types to be categorized according to their persistence and source. In ORM, the basic modeling concept is the fact, and the modeling of a domain using ORM involves identifying appropriate fact types and the roles played by different entities. Facts are classified as either static or dynamic. The latter ones are in turn classified as profiled, sensed or derived. The strengths of the graphical models is their efficiency in representing the structure of context information. As illustrated in Figure 2.2, the facts, roles and constraints annotations of ORM are extended to capture:

- Different classes of context (static facts or dynamic facts that are in turn classified as profiled, sensed or derived).
- Histories: by representing the start and end time as roles that participate in all uniqueness constraints of the fact type.
- Dependencies: by representing the relationship between facts by the binary, transitive *dependOn* relation.
- And quality: by associating facts with quality indicators such as accuracy and certainty.

Object-oriented model

This modeling approach encapsulates the representation and processes details of context entities (such as location, identity, etc.) as context objects and provides well-known interfaces to access them. The advantage of using such an approach in context modeling is to benefit from the full power of the object oriented approach (e.g. encapsulation, inheritance, reusability). User Data Convergence (UDC) [ETS11e] is such example of an object-oriented model standardized by 3GPP. This model denotes an abstract formal representation of entity type, including their properties and relationships. Furthermore, it is also used to represent the operations that can be performed on them as well as their related rules and constraints.

As illustrated in Figure 2.3 the UDC information model infrastructure is based on the Common Baseline Information Model (CBIM) [ETS11a, ET11]. CBIM describes the basic Information

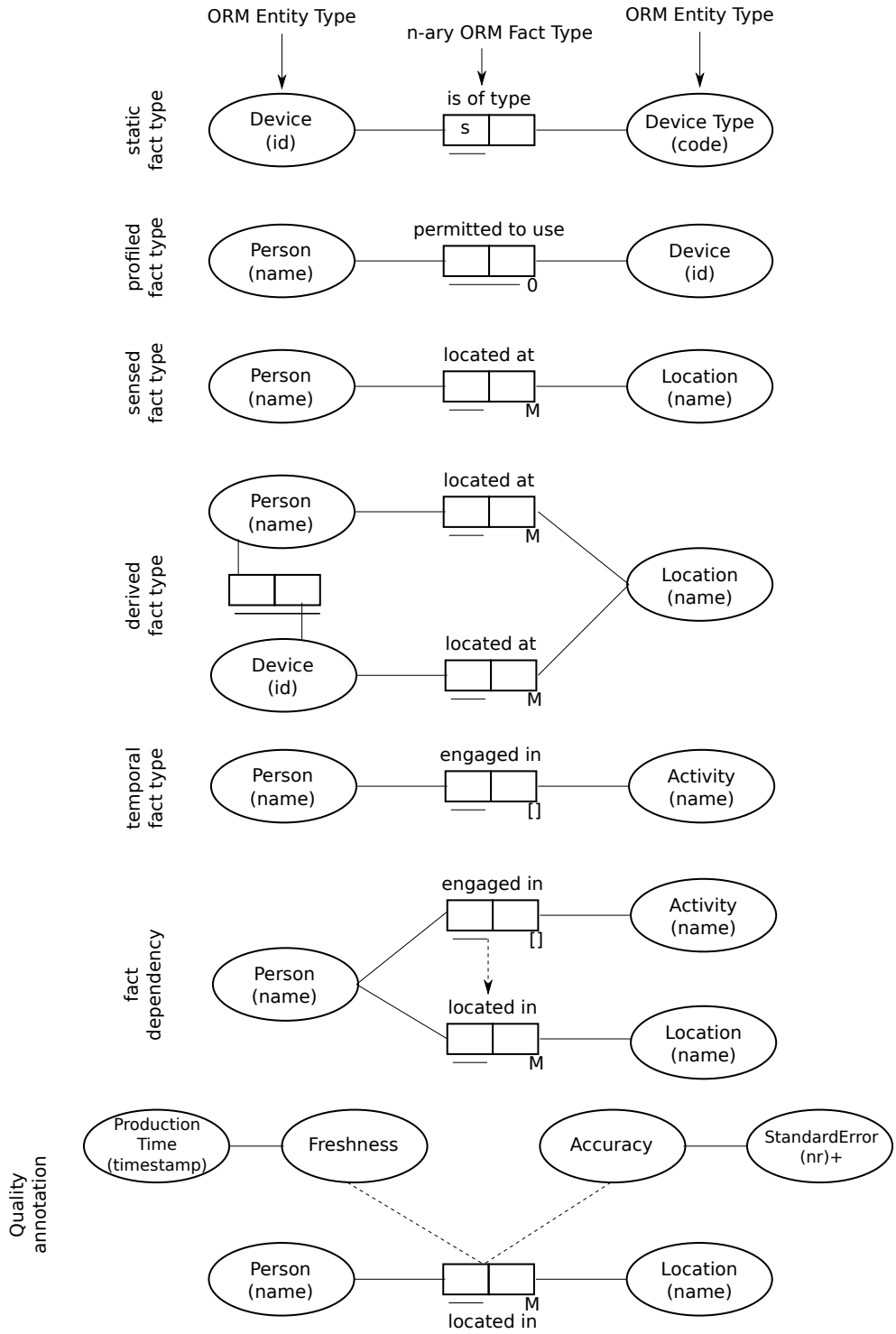

Figure 2.2: Contextual Extension ORM

Object Class (IOC) of UDC which constitutes the baseline for any given application. Moreover,

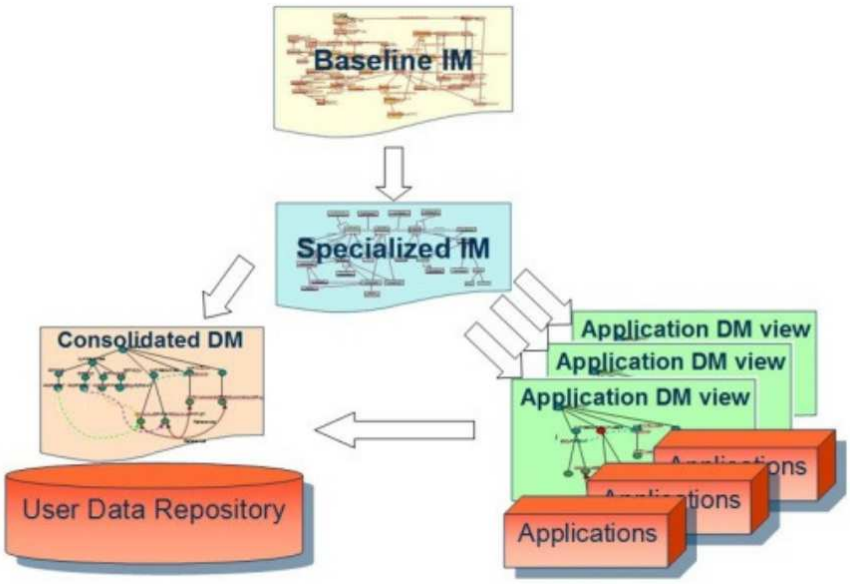

Figure 2.3: Information and Data Model of UDC

CBIM also provides support for Subscription, Service Profile, End User, Identifier, End User Group and End Device. The UML representation of the CBIM Core View and Identifiers are shown in Figure 2.4 and Figure 2.5.

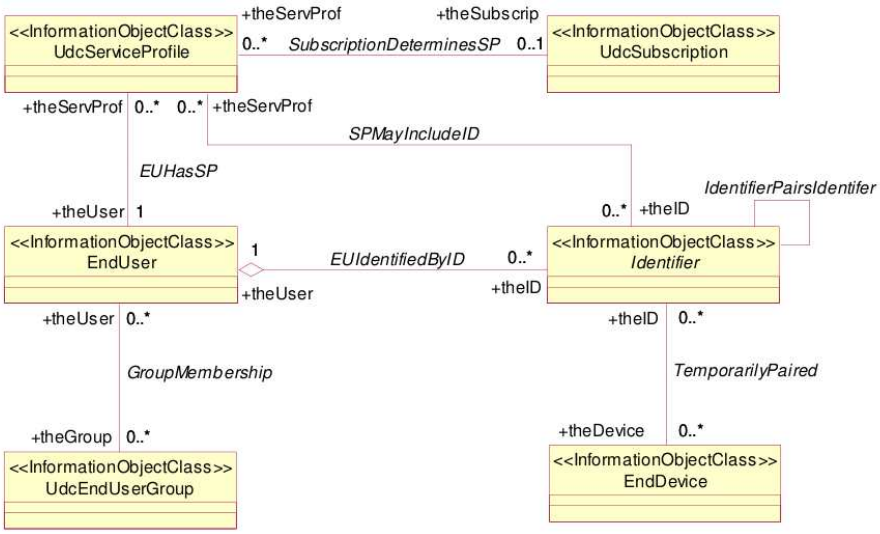

Figure 2.4: UDC CBIM Core View

Another example that uses object-oriented models is the Hydrogen project [HSP+03], their objective is to allow context sharing in a peer-to-peer manner between devices located in the same space, via WLAN, Bluetooth, etc. The Hydrogen project distinguishes between the local context (knowledge about own device) and the remote Context (knowledge about other device). Both local and remote context are modeled as context objects related to the superclass `ContextObject`. Extensibility is ensured by means of specialization. The superclass `ContextObject` is then extended by different context types such as `LocationContext` and `DeviceContext`. Each context type object has to implement the methods `toXML()` and `fromXML()` from the `ContextObject` class in order to enable context sharing through XML streams.

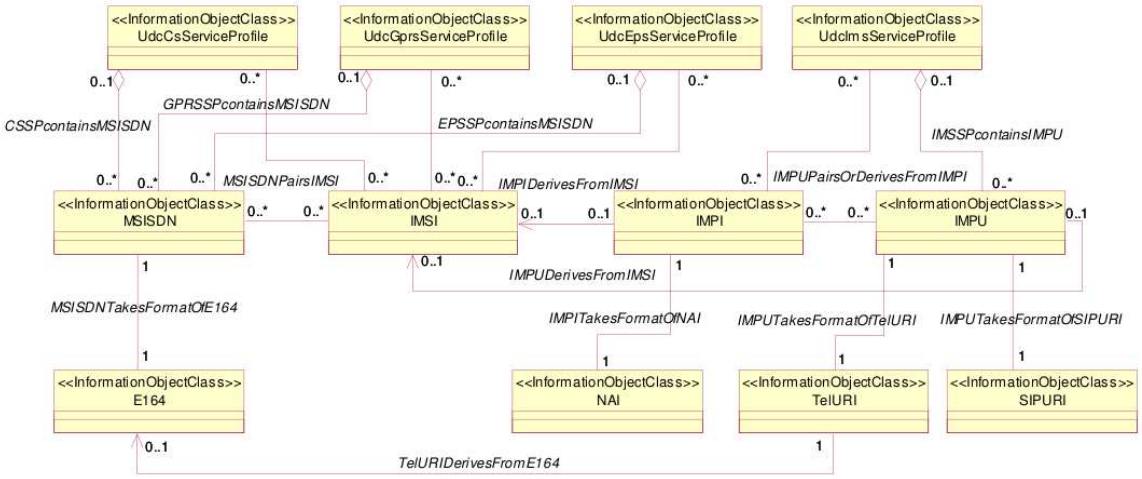

Figure 2.5: UDC CBIM - Identifiers

Logic-based model

Logic-based models define context information as facts, expressions and rules. This formal way of representation allows the model to be used for high-level reasoning and inference. Furthermore, the context information can be easily added to, modified and deleted from the logic-based system depending on changing facts derived from the system rules. Formalizing Context [McC93, MB98] was one the early works that modeled context information based on logic. The authors introduced context as abstract mathematical entities with properties useful in artificial intelligence. In fact, the aim was to allow simple axioms for common sense phenomena, e.g. axioms for static blocks world situations, to be lifted to context involving fewer assumptions, e.g. to contexts in which situations change. Further, they also mention various ways of generating new context information from old one by specialization (time or place or subject matter or context of a conversation etc.). Towards this, they use a simple assertion formula $ist(c, p)$, that asserts the proposition p is true in the context c.

Ontology-based model
Ontologies were originally defined in philosophy as the study of the nature of being, their existence or reality. Furthermore, they were also used to describe the basic categories of being and their relations. In computer science however, they are used to represent context information. Ontologies provide a way to represent formal systems information by using concepts, attributes and relations with the help of different languages namely; Ontolingua [G+93], LOOM [MB87] and Web Ontology Language (OWL) [MVH+04]. With the help of these languages ontologies has been heavily used in the following domains [GL02]

- knowledge sharing and exchange
  Ontology provide a common vocabulary which can be used by different entities.
- Logic-based reasoning and deduction
  Ontologies can be used to deduce implicit knowledge based on logic rules.
- knowledge re-use
  general usage of ontologies, such as ontologies describing temporal or spatial concepts, can be further re-used when defining an ontology for a specific domain.

The authors [SLP04], also evaluated the aforementioned context models based on a set of requirements that they believe a model should ideally have:

1. distributed composition (dc),
2. partial validation (pv),
3. richness and quality of information(qua),
4. incompleteness and ambiguity (inc),
5. level of formality (for), and
6. applicability to existing environments (app).

Distributed composition is a requirement that originates with the use of context information in ubiquitous computing. Since computing is distributed, so is the gathering and use of context. This feature is important due to lack of a central entity responsible for the creation, deployment and maintenance of data and services. *Partial validation* originates in the previous requirement. If the gathering and use of context is distributed, complete knowledge cannot be assumed for any given device, hence it should be possible to validate part of a structure. Preferably, it should be possible to support indication of *richness and quality of information*, since richness and quality of information varies over time. Further, the model should also support *incompleteness and ambiguity* of information. *Level of formality* covers the need for providing interpretations of terms used in describing context. That is, it should allow for meta-level descriptions of the terms used. Finally, *applicability to existing environments* means that it should be suitable for whatever infrastructure already exists for ubiquitous computing. The results of their evaluation is summarized in Table 2.9

Models / Requirements	distributed composition	partial validation	quality of information	incompleteness and ambiguity	level of formality	applicability
Key-Value Model	-	-	-	-	-	-
Markup scheme Model	+	++	-	-	+	++
Graphical Model	-	-	+	-	+	+
Object-oriented Model	++	+	+	+	+	+
Logic-based Model	++	-	-	-	++	-
Ontology-based Model	++	++	+	+	++	+

- absent + partially present ++ fully present

Table 2.9: Context models' appropriateness indication

2.3.1 Critical Analysis

In the above section, we briefly reviewed six context modeling approaches based on their data structure. The simplicity of the key value data structure facilitates the management of context information. Unfortunately, its lack of expressiveness prohibits any deduction from the considered context information and its flat structure does not support the relationship definition among parameters. Furthermore, the absence of data schema and meta-information on the considered context makes this type of models very difficult to reuse. Markup based models provides partial validation since they are defined by schemas. However, like the key-value models they suffer from information ambiguity, as they do not support relationship definition among parameters. The main drawback of object-oriented models is the lack of formality, since information is encapsulated. On the other hand, they provide easy distribution, allow each object to validate itself and can be integrated directly into existing object-oriented systems, although they may be heavy on resources. The strengths of the graphical models are their efficiency in representing the structure of context information. Further, they are also intuitive and easy to integrate in an UML model for the rest of the system. However, they present a low level of formalism, as they are commonly used for human structuring purpose. For logic-based models, the level of formality is extremely high and values can be distributed, but they are not suited for describing incompleteness, ambiguity, or quality of information. Towards this, the Ontology based model provides an explicit conceptualization description of data structure and semantics. Thus, ontologies are perceived as a promising tool towards adequate description and representation of context data.

2.4 Architecture for Future Media Ecosystem

The ALICANTE Project propose an open and modular architecture to support the easy creation and deployment of a networked *Media Ecosystem*. Towards this objective, it groups key actors into several domains called *environments (User, Network, Service)*. Furthermore, to facilitate cross domain interaction two novel virtual layers are considered. Utilizing the combined features of *environment* and *layers*, it aims to provide Content-awareness to the *Network Environment*; Network- and User Context -awareness to the *Service Environment*, and adapted services/content to enable enhanced QoE and better service provisioning in order to benefit all involved actors.

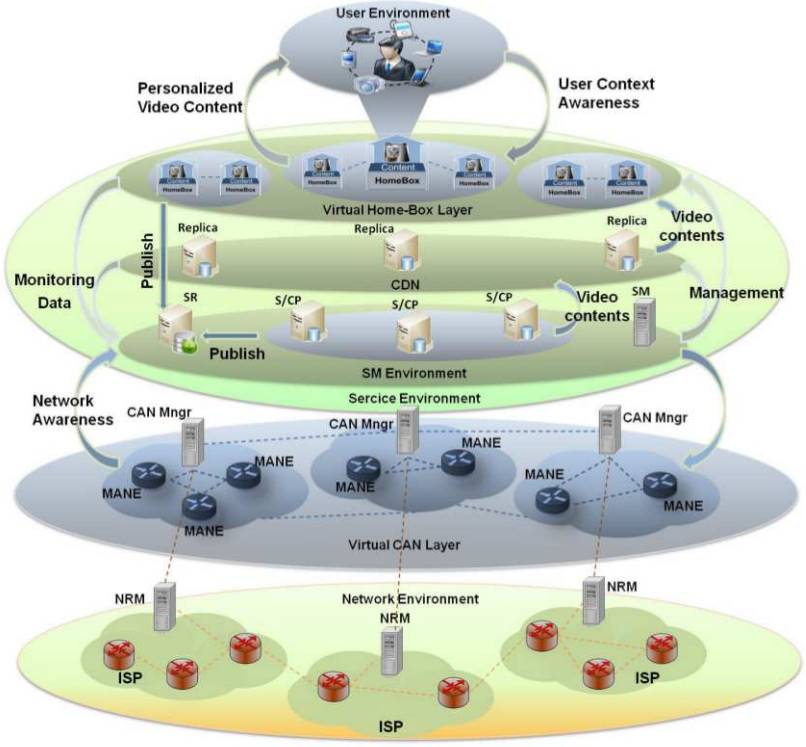

Figure 2.6: ALICANTE Architecture

The remainder of this section gives an overview of the proposed ALICANTE architecture. As illustrated in Figure 2.6 ALICANTE aims to achieve a managed media ecosystem leveraging the combined feature of *environments* and *layers*. Each of the *environments* and the two virtual layers: the Home-Box virtual layer and the Content-Aware Network layer are explained as follows:

2.4.1 The User Environment (UE)

The User Environment (UE) permits End-User (EU) to consume and/or generate content and services. It also allows the EU to access different services provided by the Service Environment. As illustrated in Figure 2.7 the EU devices connect to a virtual service called the *Terminal system*. This virtual service is hosted on the users' residential gateway (HomeBox) and helps the EU to overcome some of its limitations. The *Terminal system* enables the EU to switch between several roles, such as Content and Service Consumer, provider or Manager. However, to achieve this object the EU has to have a defined profile called the *User Profile*. The *User Profile* is made up of two parts consisting of static and dynamic parameters. The static part consists of users information including his/her likes and dislikes and other service preferences, while the dynamic part relies on QoE and QoS monitoring values.

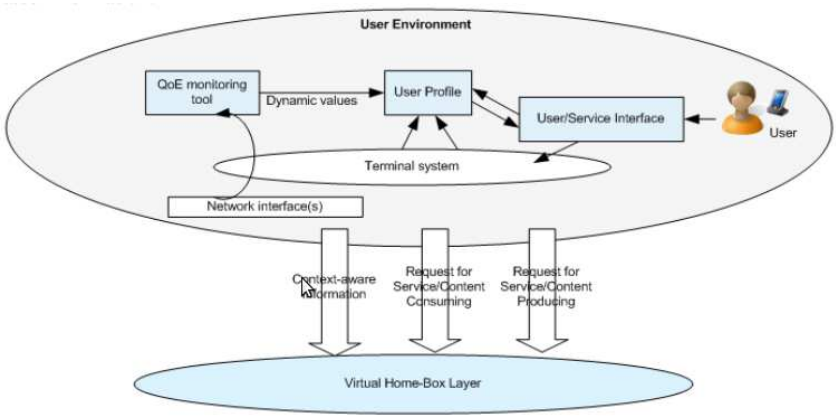

Figure 2.7: ALICANTE User Environment

2.4.2 The Service Environment (SE)

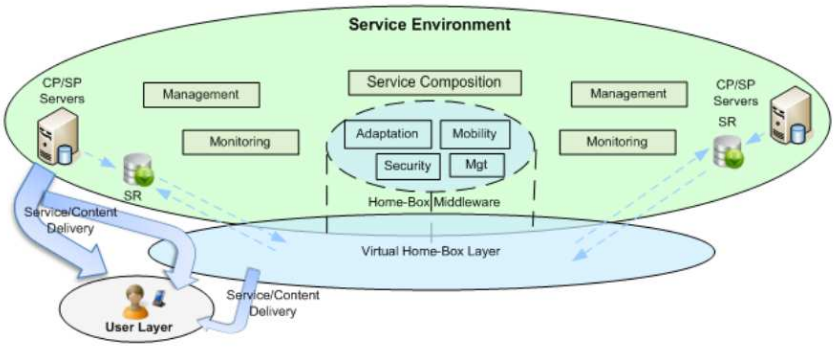

Figure 2.8: ALICANTE Service Environment

The Service Environment (SE) offers an open service provisioning platform for both EUs and Content providers [LZL+11]. Furthermore, it is also provides functionalities to manages the service life-cycle (creation, provisioning, adaptation, delivery) during its lifetime. The sole reason to promote an open service provisioning platform is to take advantage of the available context information. To this end, the SE provides context-aware and network-aware services. To enable the *-ware services the SE has distributed its functionality among two planes, namely; the *management and control plane* and the *data plane*. The *management and control plane* is responsible to gather and process context information from the EU(User Profile) and network (Monitoring tools). While, the *data plane* utilizes the context information to adapt the service/content according to EUs and network preferences. In addition to the aforementioned functionalities the SE also provides an End-to-End integrated Service management. As illustrated in figure 2.8 other functionalities supported by the SE include service composition, security and privacy.

*Content & Context

2.4.3 The Home-Box virtual Layer

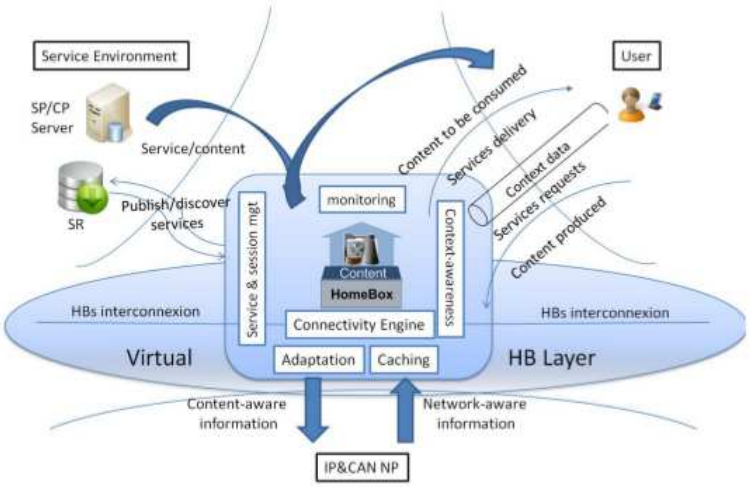

Figure 2.9: ALICANTE HomeBox

The Home-Box (HB) is a new media-centric Home Gateways having advance features like service provisioning, content caching, context management, service adaptation, content redistribution, user/service mobility and security [SCW+11]. However, to realize these advance features the HB needs to interact with two *environments*; the User and Service environment. Towards this, ALICANTE introduces a new virtual *layer* called the Home-Box layer. As illustrated in Figure 2.9 the virtual HB layer forms an interconnected overlay of HBs capable of provisioning both services and content. This virtual HB layer promotes user generated services among HBs in a flexible and optimized way. Furthermore, the HB supports unicast/multicast service delivery modes using peer-to-peer technology.

The main motivation to use P2P is to leverage the already deployed and participating HBs. These HB support disk caching, content uploading capabilities and distribution of popular content just like existing Content Delivery Networks (CDN)s. Furthermore, they also provide scalability, reliability and improve the service responsiveness and content availability while saving network bandwidth and ensuring low cost delivery [CNCS12]. The proposed architecture keeps the high control of managed networks while taking advantage of the self-scaling property of P2P solutions. The HBs are managed by the SE and the Service Manager (SM) acts as a P2P bootstrap server and the Service Registry as a P2P tracker for the HBs peers. This enables the HBs to allow the End-Users to compose and publish and distribute their contents and services.

2.4.4 The Network Environment (NE) and the CAN Layer

The Network Environment (NE) is the backbone of the architecture providing connectivity among

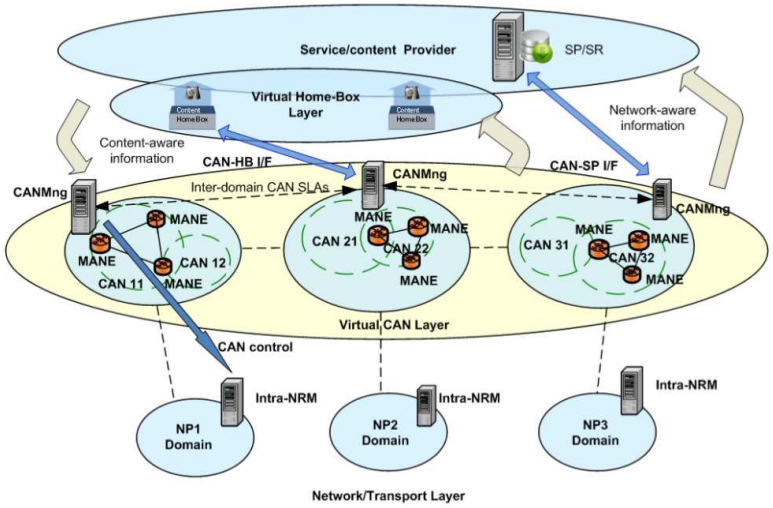

Figure 2.10: ALICANTE Network Environment

other *environments*. To promote customized facilities to all Network Provider (NP) the ALICANTE consortium proposed a rich new virtualized network space. This virtualized network space is made up of two *layers* namely; the Content-Aware Network (CAN) *layer* and the Network *layer*. As depicted in Figure 2.10 together these *layers* help NP overcome some of its limitations.

The *CAN layer* is in charge of providing customized QoS to actors on top of the IP infrastructure. Furthermore, it is entitled to construct a mono-domain or multi-domain Virtual CAN (VCAN). One or several VCANs can be constructed having different capabilities and installed in each domain. The VCANs are virtual networks that support enhanced packet/flow processing in core network nodes [BNT10]. Their use have several advantages like, improving data delivery via content-aware traffic classification and advance processing (filtering, routing, forwarding, QoS-processing, adaptation-dynamic, aggregated or per flow, security, monitoring). The CAN layer is managed by dedicated CAN Managers. Each network domain has one CAN Manager, this ensures seamless deployment of ALICANTE system in real world.

The *Network layer* is responsible for initiating the CANs. Each CAN Manager takes the help of its Intra domain Network Resource Managers (Intra-NRM) to initiates their respective CANs. Furthermore, ALICANTE introduces a new network to support its *-ware feature called Media-Aware Network Element (MANE) [NSV+11]. The MANEs are the real content-ware nodes in the architecture. They process the flow according to the content properties and network current status: content-aware intelligent routing and forwarding, flow adaptation, QoS and resource allocation, filtering and specific security functions, and possibly data caching.

*Content & Context

2.4.5 Critical Analysis

In the above section, we presented a high level description of ALICANTE Project which proposes, a networked media ecosystem for FI. For the research work presented in this thesis, the focus is put on the User and Service Environments to enable ❶ service discovery in the highly heterogeneous networks and ❷ to provide high quality services and personalization towards a better experience for the End User. The User Environment presented some advantages for using the *User Profile* by allowing users to switch between different roles. It, however, does not allow them to discover and publish services using protocols other then the once used by the *Terminal system*. Nevertheless, End-Users should be able to discover and publish services via the *Terminal system* irrespective of the protocol they use. On the other hand, the ability of ALICANTE SE to separate *data* and *management* functionality gives it a unique advantage to enable the two functionalities to evolve separately. Furthermore, with the help of Service Registry (SR) it provides a large scale service infrastructure. However, this does not mean that it can interact and request services from external sources(YouTube, google+ etc) as well as existing platforms like IP Multimedia Subsystem (IMS) and Internet Protocol Television (IPTV).

Need for Interoperability
Due to exponential growth in networked applications and distributed resource sharing, there is a strong incentive for an open large-scale service provisioning infrastructure operating over heterogeneous networks. However, service globalization may require interoperation of diverse discovery systems based in independently administrated domains. Part of that interoperation is sharing information about deployed services across domain boundaries. SDPs like SLP [GPVD99], UPnP [PFK+08] and Bonjour [Dro97] do not cross domain boundaries, while those that rely on P2P technologies [MM02, SMK+01, RFH+01] are not interoperable with each other. Nevertheless, to achieve service globalization, a service must be identified during its life cycle independently of the network it belongs to at a given time. Therefore, there is a need for an interoperable discovery system that can incorporate multiple heterogeneous discovery approaches. In Chapter 3.1, we review some of the existing interoperability solutions and highlight their advantages and drawbacks.

Need for context-awareness
Over the last few years, the focus has been progressively put on user satisfaction. This is due to the fact that, for the user only the service value counts, not the network or the software on the device. Towards this objective, context information plays a very important role by elevating the QoS towards EU's expectations. Context covers a wide range of heterogeneous information that continues to grow with the advent of new services and technologies. It is therefore useful to uniquely classify and organize this information so that relevant entities can utilize it to provide better QoS to EU. Based on its characteristics, context information is classified as *static* and *dynamic*. *Dynamic* context information is the one that changes at a rapid rate, while the *static* context information is more stable over a longer period. This does not necessarily mean that the latter ones never change but they do not change as fast as the dynamic ones. This classification is mainly used in time critical systems like media services where network and devices

characteristics may change any time. Thus, it becomes necessary to model context information and to address it from management perspective. Section 2.3 showed the importance of ontology to model context information. In Chapter 3.2, we review some of the existing ontology-based context modeling solutions and highlight their contributions.

2.5 Summary

In this chapter, we have highlighted the background of our work by highlighting the importance of service discovery. To facilitate the discussion further, we have separated the background into three parts. In the first part, we have presented popular service discovery implementations and classified them as SDPs for either local area network or large scale networks. Then, we have reviewed each of them highlighting their standard features and identifying their key limitations. In the second part, we have briefly reviewed popular context models. We have also shown how ontology based models are a good solution for modeling context information in ubiquitous computing. In the third part, we have described an architecture for future media ecosystem, ALICANTE. And finally, we identified some limitations in the User and Service Environment that need to be addressed in order to enable highly personalized services in large scale heterogeneous networks. In Chapter 3, we present the current state of the art addressing protocol interoperability and ontology based context modeling.

Chapter 3

Related Work

"Dont waste your time with explanations: people only hear what they want to hear"

–Paulo Coelho

Contents

3.1 Existing Interoperability Solutions

3.1.1 ReMMoC

ReMMoC [GBS03]green is a configurable and dynamically reconfigurable reflective middleware that supports mobile application development and overcomes the heterogeneous properties of

the mobile environment. ReMMoC uses OpenCOM as its underlying component technology and it is built as a set of component frameworks. Utilizing the different components from OpenORB the frameworks increases the size of the middleware implementation. However using extra management functionality for managing reconfiguration could exhausts the constrained resources of a mobile device. Therefore, as depicted in Figure 3.2 ReMMoC consists of only two component frameworks: ❶ a binding framework for interoperation with mobile services implemented upon different middleware types, and ❷ a service discovery framework for discovering services advertised by a range of service discovery protocols. The binding framework is configured by plugging in different binding type implementations and the service discovery framework is similarly configured by plugging in different service discovery protocols. Adding more component frameworks for other non-functional properties such as security and resource management can extend the platform at a later stage. In addition to the aforementioned components ReMMoC also provides a generic Application Programming Interface (API) to develop mobile applications.

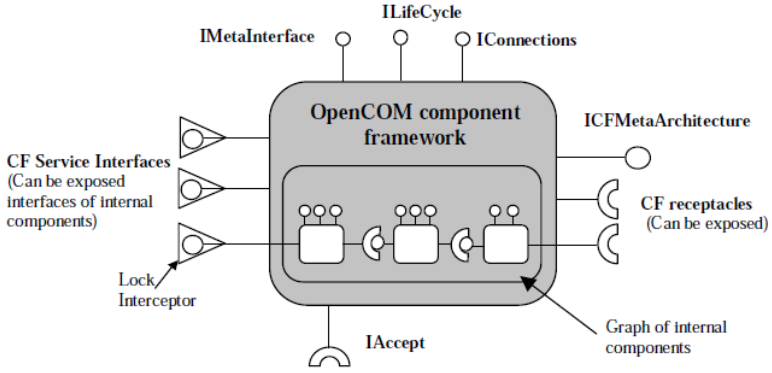

Figure 3.1: OpenCom Architecture

OpenCOM Component Frameworks: A Component Framework (CF) is defined as a collection of rules and contracts that govern the interaction of a set of components [SGM02]. The motivation behind component frameworks is to constrain the design space and the scope for evolution. A component framework in OpenCOM is itself an OpenCOM component that maintains internal structure (a configuration of components) to implement its service functionality. The design of these component frameworks is based upon the concepts of composite components proposed by OpenORB [BCA+01]. Therefore, component frameworks can be composed, replaced and connected together in the same manner as components. To provide this capability, as illustrated in 3.1 each OpenCOM CF implements the base interfaces of an OpenCOM component (*IMetaInterface*,*ILifeCycle*, *IConnections*) in addition to its own interfaces and receptacles. The interfaces and receptacles of internal components can be exposed to create these.

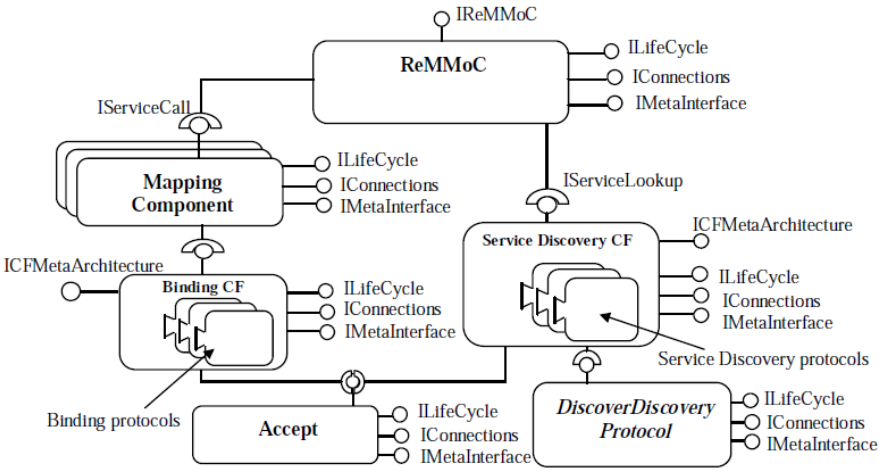

Figure 3.2: ReMMoc Architecture

The Binding Component Framework: The primary function of the binding framework is to interoperate with heterogeneous mobile services. Therefore, over time it may be configured to make a number requests, or change to a subscribe configuration and wait to receive events of interest. Different middleware paradigms, synchronous or asynchronous (e.g. tuple spaces, media streams, Remote Procedure Call (RPC), publish-subscribe or messaging), can be plugged into the binding framework if they have been implemented using `OpenCOM` components. Within the binding framework changes are made at two distinct levels. Firstly, each binding type implementation can be replaced by a publish-subscribe subscriber. This dynamic reconfiguration is performed by receiving information from the service discovery framework describing the type of binding. Secondly, fine-grained changes to each configuration can be made in light of environmental context changes, such as those involving quality of service, or changes in the applications requirements. For example, an application may require server side functionality, in addition to the existing client side; therefore components implementing server side functionality are added.

The Service Discovery Framework: The Service Discovery framework allows services that have been advertised by different service discovery protocols to be found. The framework is configured to discover protocols currently in use in the environment. For example, if `SLP` is in use, the framework configures itself to an `SLP` Lookup personality. However, if `SLP` and `UPnP` are found then the frameworks configuration will include component implementations to discover both. Like the Binding CF, fine-grained component changes can be made. For example, in `SLP` you may wish to perform lookup using just the multicast protocol if no directory agent is present, but at a later stage if a directory agent is discovered the configuration can be changed to direct requests to it. The service discovery framework offers a set of generic service discovery methods through the *IServiceLookup* interface. This includes a generic service lookup operation that returns the information from different service discovery protocol searches in a

generic format. For example, a lookup of a weather service across two discovery configurations, e.g. UPnP and SLP, returns a list of matched services from both types. It is this information (the description of the service returned by the lookup protocol) that is used to configure the binding framework. Initially, the discovery protocol(s) that are currently in use at a location must be determined. The *DiscoverDiscoveryProtocol* component, which is plugged into the framework, tests if individual service discovery protocols are in use, either upon a synchronous request or by continuously monitoring the environment and generating an event on detection. Continuous monitoring will quickly use up resources (e.g. battery power); therefore in some cases synchronous checking may be appropriate. The service discovery framework utilizes this behavior to automatically reconfigure itself. Other methods for discovering discovery protocols, not currently included in the implementation, may utilize the devices context information, e.g. if the device is currently using a Bluetooth connection then an SDP personality is configured. Furthermore, the middleware may use prior knowledge to select an appropriate protocol, i.e. the platform stores context information per location that details which service discovery protocols were used at that point previously.

```
interface ReMMoc_ICF : IUnknown {
   HRESULT WSDLGet (WSDLService* ServiceDescription, char* XML);

   HRESULT FindandInvokeOperation (WSDLService ServiceDescription, char*
                        OperationName, int Iterations, ReMMoCOPHandler Handler);

   HRESULT InvokeOperation (WSDLService ServiceDescription, ServiceReturnEvent
      ReturnedLookupEvent, char* OperationName, int Iterations, ReMMoCOPHandler Handler);

   HRESULT CreateOperation (WSDLService ServiceDescription, ServiceReturnEvent
      ReturnedLookupEvent, char* OperationName, int Iterations, ReMMoCOPHandler Handler);

   HRESULT AddMessageValue (WSDLService *ServiceDescription, char* OperationName,
                  char* ElementName, ReMMoC_TYPE type, char* direction, VARIANT value);

   HRESULT GetMessageValue (WSDLService *ServiceDescription, char* OperationName,
                  char* ElementName, ReMMoC_TYPE type, char* direction, VARIANT value);
}
```

Figure 3.3: ReMMoc API

ReMMoc API: ReMMoC programming model is based upon the concept of Web Service Description Language (WSDL) described abstract services. Application developers must utilize these WSDL definitions in the style of Interface Definition Language (IDL) programming. To maintain a consistent information flow to the application an event-based programming model, that overrides the different computational models of each paradigm, is offered. Each abstract service operation is carried out and its result is returned as an event. For example, if that operation is executed by an Remote Method Invocation (RMI) or an event subscription the result is always an event. Similarly, service lookup operations return results as events. Figure 3.3 documents the API of ReMMoC, which consists of operations to: lookup services, lookup then invoke abstract WSDL operations, invoke operations on known services, or create and host service provider operations. ReMMoC maps these API calls to the binding framework through the use of a reconfigurable mapping component.

3.1.2 INDISS

Interoperable Discovery System for Networked Services (INDISS) [BI05] is a transparent solution to overcome SDPs heterogeneity. Its main objective is to minimizing resource usage (i.e., memory, processing and bandwidth), and introduces lightweight mechanisms that may be adapted easily to any platform. `INDISS` is composed of a set of event-based components and their composition is performed dynamically at run-time according to both the context and the device on which `INDISS` is deployed. `INDISS` operates close to the network, capturing and translating network messages without the need for clients or services interactions. As a result, service discovery interoperability is provided to applications without altering them. As depicted in Figure 3.4 INDISS consists of three main components, namely ❶ the Monitor, ❷ the parser and ❸ the composer.

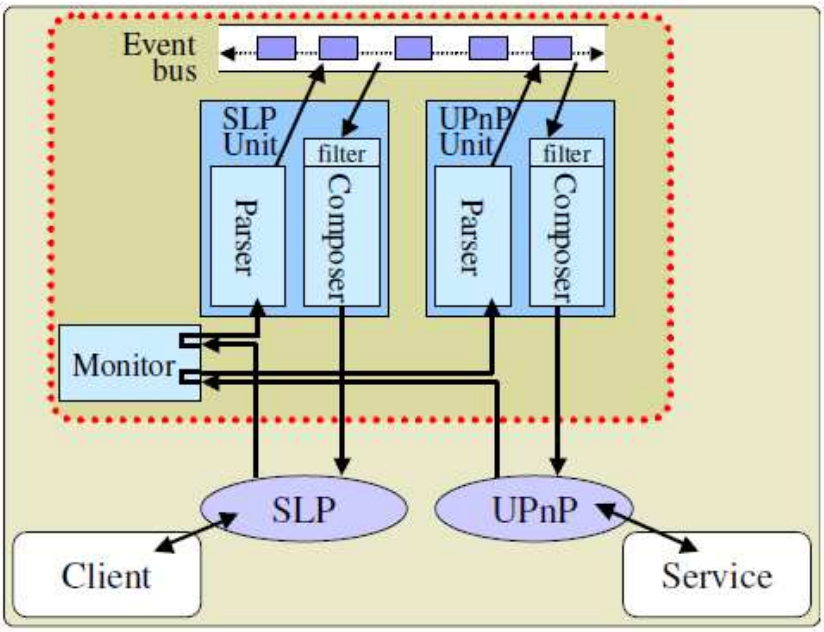

Figure 3.4: INDISS Architecture

Monitor Component: The monitor component detects the SDPs that are used based on network activity on the assigned multicast groups and ports. As depicted in Figure 3.5 this component also captures/collects network messages sent by clients and services onto these multicast groups, and forwards them to the appropriate parser components.

Parser Component: The parser component, associated to a specific SDP, transforms the raw data flow (i.e., network messages) into series of events, extracting semantic SDP concepts from

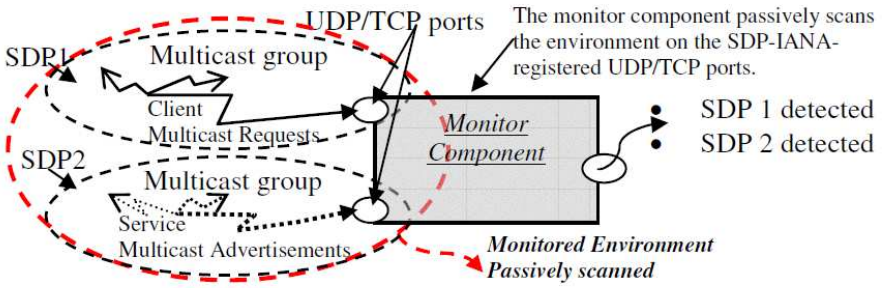

Figure 3.5: INDISS: Monitor Component

syntactic details of the SDP messages. The generated events are delivered to an event bus locally deployed.

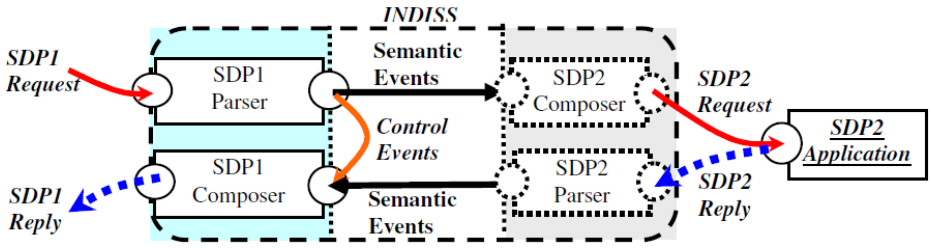

Figure 3.6: INDISS: Parser Composer Component

Composer Component: The composer component delivers a SDP message understood by the target clients and/or services based on specific sets of events received from the event bus. Figure 3.6 highlights the interaction that takes place among different parser composer components.

Parsers and composers are dedicated to specific SDP protocols. In `INDISS`, the communication between the parser and the composer does not depend on any syntactic detail of any protocol. They communicate at a semantic level through the use of events. A fixed set of common events has been identified for all SDPs, and each SDP has also a set of specific events. SDP interoperability comes from the composition of multiple parsers and composers dedicated to different SDP, and the implicit creation of an event bus.

Event-based Coordination The overall coordination process in `INDISS` is implemented by a event-based unit called *SDP*. The *SDP* unit is specified using a Finite State Machine (FSM).

A *SDP* state machine is a graph of states connected by transitions. This state machine is a Deterministic Finite Automata (DFA) and is defined as a 5-tuple $(Q, \sum, C, T, q0, F)$, where Q is a finite set of states,$\sum$ is the alphabet defining the set of input events (or triggers) the automaton operates on, C is a finite set of conditions, $T : Q \times \sum \times C \rightarrow Q$ is the transition function, $q0 \in Q$ is the starting state and $F \subset Q$ is a set of accepting states. States keep track of the progress of the SDP coordination process. Transitions are labeled with events, conditions and actions. The occurrence of an event may cause a transition if the event matches both the event and the condition of the transition. When a transition is engaged, several actions may be executed, relating to translation of events to/from message data, coordination, and configuration management. In `INDISS` the *SDP* DFA is dedicated to one protocol to account for the protocol's specifics and consequently realize some optimization. Events are basic elements and consist of two parts: *event type* and *data*. Whatever their types, events are always considered as triggers for the unit components to react and eventually activate some coordination rule. We define the minimal/ mandatory set of events that is common to all SDPs and sets of specialized events that are specific to SDPs. The set of mandatory events $\sum$ is defined as the union of a number of subsets.

3.1.3 Z2Z

`z2z` [BRLM09] is a protocol translator. It uses a generative approach to construct network gateways based on Domain Specific Language (DSL). `z2z` uses three DSL namely; ❶ Protocol Specification Language (PSL), ❷ Message Specification Language (MSL) and ❸ Message Translation Language (MTL). These DSLs relies on advanced compilation strategies to hide complex issues from the gateway developer such as asynchronous message responses and the management of dynamically-allocated memory, while remaining in a low-overhead C-based framework. Additionally, it also provides a runtime system that addresses a range of protocol requirements, such as unicast vs. multicast transmission, association of responses to previous requests, and management of sessions. Figure 3.7 highlights the z2z architecture

Protocol Specification Language
The protocol specification language defines the properties of a protocol that a gateway should use when sending or receiving requests or responses. The PSL enable developers to declare the following as information block, namely; ❶ *Attributes*, ❷ *Request*, ❸ *Sending* and ❹ *Flow and session flow*. The *attributes* block of the protocol specification language indicates what characteristics (unicast, multicast, synchronous or asynchronous) are used by the protocol to interact with the network. Based on this information, the z2z runtime system provides appropriate services. The *request* block of the protocol specification language declares how to map messages to handlers. While the *sending* block defines the mandatory basic information needed by all messages(request and response). Furthermore, it allows developers to declare local variables which can used to store relevant information over the treatment of all messages. And finally, the *Flow and session flow* block keeps tracks of the association of request and responses with their session.

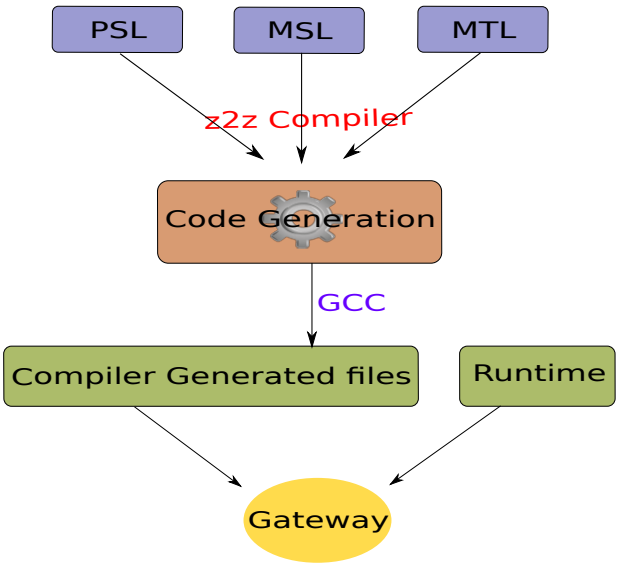

Figure 3.7: Z2Z Architecture

Message Specification Language
The message specification language is used to describe the messages that can be received and created by a gateway. Based on this description, the `z2z` compiler generates code for accessing message elements and inserting message elements into a created message. There is one message specification per protocol relevant to the gateway, including both the source and target protocols, as represented by the protocol specification language, and one per any higher level message type that can be embedded in the requests and responses. The message specification language provides a *message view* describing the relevant elements of incoming messages and `templates` for creating new messages. The set of elements is typically specific to the purpose of the gateway, not generic to the protocol.

Message Translation Language
The message translation language expresses the message translation logic, which is the heart of the gateway. It provides a set of handlers, one for each kind of relevant incoming request, as indicated by the protocol specification module. Handlers are written using a C-like notation augmented with domain-specific operators for manipulating and constructing messages, for sending requests and returning responses, and for session management.

3.1.4 Starlink

`Starlink` framework [BGR11] utilizes high-level models of each individual protocol to generate interoperability bridges at runtime. `Starlink` provides a set of DSL to specify these models, whose content forms the overall interoperability logic. The `Starlink` framework, when deployed in the network, executes this logic transparently from the protocols and ensures that two legacy systems can interoperate dynamically. As depicted in Figure 3.8 the `Starlink`

framework is made of three main components namely; ❶ the *network engine*, ❷ the *message parsers and composers* and the ❸ *automate engine* which uses the concept of k-Colored automata to capture the protocol behavior.

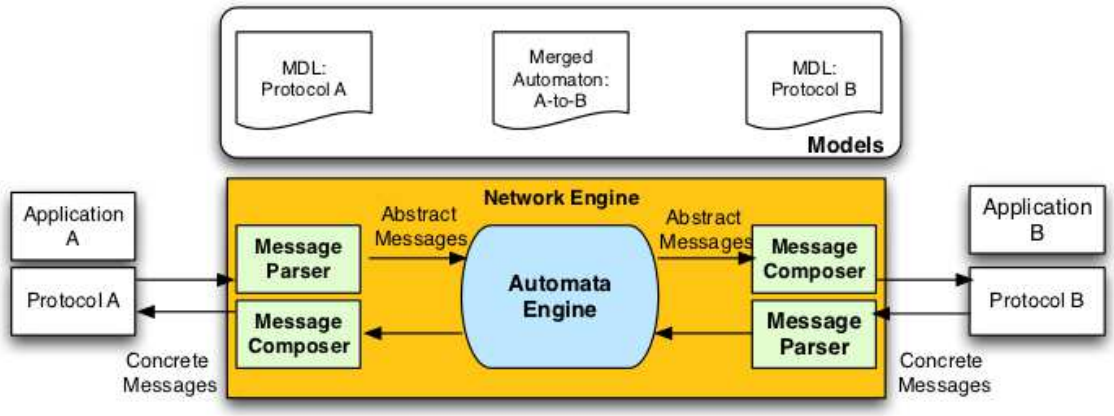

Figure 3.8: Starlink Architecture

network engine
sends and receives physical messages (i.e. data packets) to and from the network. A transition in the *k-colored* automata attaches network semantics to describe the requirements of the network. The network engine then provides the services to meet these requirements, which could include different types of transport or multicast behavior. The current implementation of the network engine provides traditional TCP and UDP services for infrastructure networks. However, the architecture is configurable so that if Starlink were to be deployed in more heterogeneous environments, e.g. ad-hoc networks, this network engine could be replaced with configurable services for ad-hoc routing [RGCH09].

message parsers
read the contents of a network packet and parse them into the *AbstractMessage* representation such that the data can be manipulated during the mediation process. For example, if a HTTP message is received a HTTP parser reads all the fields of the header and body. Correspondingly, message composers construct the data packet for a particular protocol message, e.g. constructing the content for a HTTP GET message. Importantly, the message composers and parsers are generic reusable software elements that interpret high-level specifications of message content. The Message Description Language (MDL) specification utilize these generic components at runtime to create a specific protocol parser or composer.

automata engine
executes the behavior of the merged automata, i.e. it controls the sequence of sending, receiving,

parsing, composing and translation of messages. In Starlink, there a three types of states: ❶ a receiving state waits to receive a message and will only follow a matching receive transition when a matching message is received; ❷ a sending state sends a message described in the single transition; ❸ a no-action state is a translation state that translates data from the fields on one or more of the prior messages into the message to be constructed.

k-Colored Automata

The Starlink framework capture the behavior of protocols by using a k-colored automaton $A_k = (Q, M, q0, F, Act, \rightarrow, \Rightarrow)$, where Q is a finite set of states, M are either incoming or outgoing abstract messages, $q0 \in Q$ is the starting state and $F \subset Q$ is a set of accepting states. Act is a set of actions such that $Act = ?, !$ where $?$ is the receive action and $!$ is the send action. $\rightarrow \subseteq Q \times Act \times M \times Q$ is the transition relation that can be either a *receive-transition* or a *send-transition*. The former has the following form $s_1 \xrightarrow{?m} s_2$ for $(s_1, ?, m, s_2) \in \rightarrow$ and changes the state of the automaton from s_1 to s_2 once the message m is received. The latter is noted $s_1 \xrightarrow{!m} s_2$ for $(s_1, !, m, s_2) \in \rightarrow$ and indicates the next state to which the automata passes as soon as the message m is sent. Moreover, each state maintains a queue to store both incoming and outgoing message instances. A sequence of stored messages is represented by a message vector noted $\vec{m}$. By either $s_i.m$ or $si.\vec{m}$, Starlink denote a particular stored message or a sequence of stored messages from a specific state s_i. To further analyze at runtime the behavior of an automaton, Starlink define a history operator as follows $\Rightarrow \subseteq Q \times Act \times \vec{m} \times Q$. Thus, let $s_1, s_2 \in Q$, $s_1 \overset{!\vec{m}}{\Rightarrow} s_2$ (resp. $s_1 \overset{?\vec{m}}{\Rightarrow} s_2$) gives the sequence of the sent (resp. received) instances for each abstract message from the state s_1 to s_2. In order to capture these low level network semantics, Starlink use automaton coloring which consists of assigning labels called colors to states of the automaton. An automaton can pass successfully from one state to another, following either a *receive-transition* or a *send-transition*, without any network issues, only if the concerned states share the same color. An automaton A_k is said to be k-colored if all its states are k-colored, and if there exists a function f such as $f(h(key1; val1); (key2; val2); ...; (keyn; valn)i) = k$. Function f is a perfect hash function that maps a list of tuples, where each tuple is a key-value pair describing low level network details, to a unique hash value k (i.e. without collisions).

3.1.5 Critical Analysis

In the above section, we reviewed some of the existing protocol interoperability solutions. ReMMoC is a configurable and dynamically reconfigurable reflective middleware. It brings forward the concept of components and interfaces proposed by *OpenORB*. These components and interfaces enable the middleware to add new functionalities to ReMMoc. Furthermore, it also provides a generic API in order to help developers provide interoperable solutions. However, ReMMoC require developers to redesign all existing applications to make them compliant with the ReMMoC API, which is quite a daunting task. This particular constraint is overcome with INDISS which is a transparent middleware that provides interoperability to existing applications without altering them. INDISS replies of event-based components. However, extending INDISS to support new protocols is a challenging task as it requires both a deep knowledge of the protocols involved,

and also a substantial understanding of low-level network programming. Although ReMMoC and INDISS could be considered as a step forward in the challenge of interoperable service discovery. z2z and Starlink have brought forward many facilities to enable transparent translation of one protocol to another. z2z uses a generative approach to enable protocol translation, while starlink relies on k-colored automata. Furthermore, they provide an optimized runtime system and facilities for describing network protocol behaviors, message structures, and translation logics. Such facilities come from the fact that they rely on a high-level definition language that hides low level network details and highlights only key properties needed for protocol translation. In our solution, we rely on the concept of components introduces by ReMMoC, the event-based network monitoring brought forward by INDISS and facilities provided by z2z as a transparent protocol translation component to provide an interoperable discovery solution for Future Internet.

3.2 Existing Ontology-based Solutions

3.2.1 CoOL

Context Ontology Language (CoOL) [SLPF03] is an Aspect-Scale-Context (ASC) model which can be used to enable context-awareness and contextual interoperability during service discovery. The main component is a *reasoner* (also called the *inference engine*) which infers conclusions about the context based on an ontology built with CoOL. Ontologies offers flexibility and extensibility we need in distributed systems, as they can be stored at different places and created by different authors. Figure 3.9 shows the integration of the CoOL architecture in the context provider domain. *OntoBroker* system [DEFS98], is used as the *reasoner* as it supports F−Logic [KLW95] a knowledge representation and knowledge query language. F−Logic is a logic language combining object-oriented and predicate logic characteristics not based on XML. It is more expressive than OWL and is much more appropriate for specifying relevance conditions.

Aspect-Scale-Context (ASC) Model
Aspect-Scale-Context (ASC) model is named after the core concepts of the model, which are *aspect*, *scale* and *context information*, see Figure 3.10. Each *aspect* aggregates one or more *scales*, and each *scale* aggregates one or more *context information*. These core concepts are interrelated via *hasAspect*, *hasScale* and *constructedBy* relations. A *scale* is an unordered set of objects defining the range of valid *context information*. In other words, a valid *context information* with respect to an *aspect* is one of the elements of the aspect's *scales*.
On an abstract level, *context information* may be seen as content data complemented by some meta data characterizing the content data. Each *context information* has an associated *scale* defining the range of valid instances of that type of *context information*. Context information characterizing the content of another context information is a meta information and thus a context information of higher order and expresses the quality of the lower order context information.All scales within one aspect are constrained by the ASC model in a way, that there exist a mapping

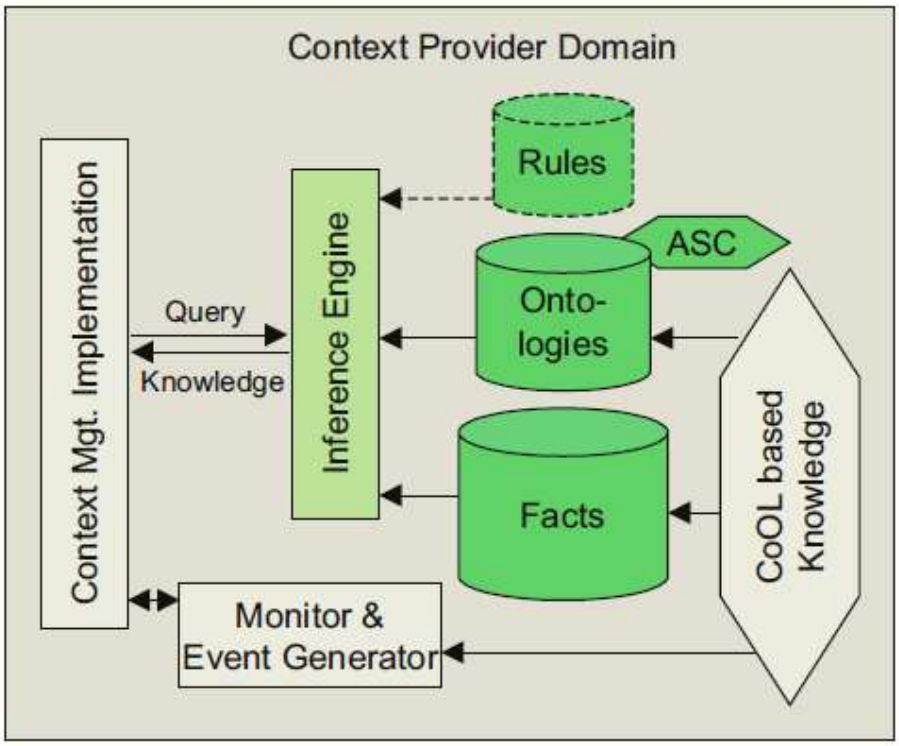

Figure 3.9: CoOL Architecture

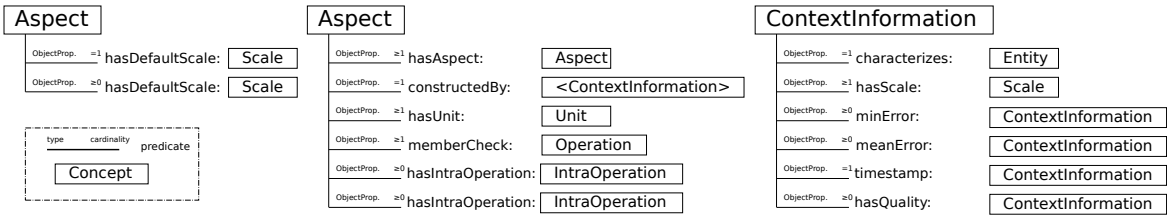

Figure 3.10: CoOL: Aspect-Scale-Context Model

function from one scale to at least one other of the already existing scales of the same aspect. This function is called IntraOperation, see Figure 3.11 Scales which require access to scales of one or more other aspects can be defined using InterOperations, Due to the fact that each scale is an unordered set of context information instance objects, there may be no relative sort order between the context information inherently given. Therefore they introduced the MetricOperation which may be used to compare two context information instance objects of the same scale in an implementation-defined manner to see if they match or what their relative sort order is by returning either the first or the second parameter. Thus the return value indicates the ordering of the two objects. Information about the signature of any InterOperation, IntraOperation or MetricOperation is available in the signature specification pointed to with the property identifiedBy.

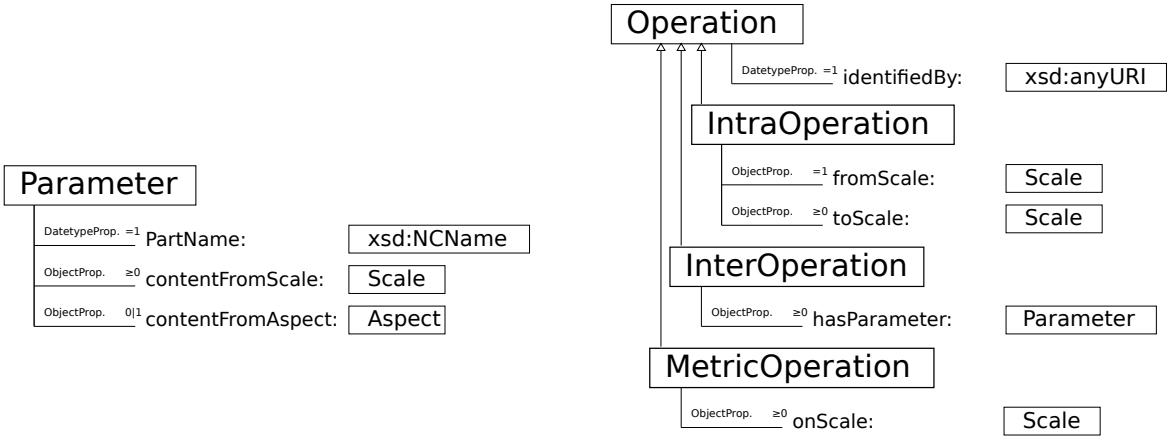

Figure 3.11: CoOL: Aspect-Scale-Context Operation

3.2.2 COBRA-ONT

COBRA−ONT [CFJ03]is a collection of ontologies, expressed in OWL, defined in the frame of *Context Brocker Architecture*, an architecture/framework designed to support context-sensitive pervasive systems. COBRA−ONT has been designed for the context representation of a smart meeting-room and consists in four sub-ontologies: *Place*, *Agent*, *Agents's Location* and *Agent's Activity*. Concepts defined in COBRA−ONT, as well as related properties and relations between these concepts.
The top level class in COBRA−ONT is *Place*, which represents an abstraction of the physical location of the modeled smart space. A location is described with a name, a longitude and latitude. The Place concept is the union of 2 concepts: *AtomicPlace* and *CompoundPlace*. A location may include other locations. For instance, the Campus and Building locations include the Room, Hallway and Stairway locations. The notion of capacity is represented by the *spatiallySubsumes* and *isSpatiallySubsumedBy* relations. The ontology presenting the agents and able to act on the system is built around the *Agent* concept which inherits from two distinct concepts: *Person* and *SoftwareAgent*. An agent can be described by different attributes such as its name or email address. Roles, such as *SpeakerAgent* or *AudienceAgent*, are assigned to agents, through the *fillsRole* property. In order to deduce actions that an End-User intends to, the relation *intendstoPerform* of the Role concept is defined and takes as value an instance from the *IntentionalAction* class. The ontology *Agent's Location* contains the basis for dynamic knowledge that allow describing the localization of an agent. To do so, the *LocatedIn* relation is added to the Agent concept. This relation points on Place concepts. Considering that a location can be specialized in *AtomicPlace* and *CompoundPlace*, the following axioms are defined:

- No agent can be present at two *AtomicPlace* locations within the same timespan ;
- And one agent can be present at two different *CompoundPlace* locations within the same timespan only if one location includes the other. This type of reasoning process is important for inconsistency detection when it comes to find/know the location of the agent.

It is also possible to build an agent-focused classification based on their location: *PersonInBuilding* and/or *SoftwareAgentInBuilding*. Furthermore, the other related classes are *PersonNotInBuilding* and *SoftwareAgentNotInBuilding*. Relying on OWL, COBRA-ONT can therefore propose reasoning options on OWL semantics. It is also possible to integrate in language rules specific to the domain. This not only enable the detection and solving of inconsistencies, but also the interpretation of sensors-transmitted context information.

3.2.3 SOUPA

SOUPA (Standard ontology for OWL-based Ubiquitous and Pervasive Applications)[CPFJ04], was proposed by the authors of COBRA-ONT as an enhancement to address interoperability. To realize this aim, they used external vocabularies like Friend-Of-A-Friend ontology (FOAF) [BM07] [PJYF03], DAMLTime [PH04], MoGATU BDI ontology [Per04] and Rei policy ontology [KFJ03]. Nevertheless, to provide a common ontology to different pervasive applications, SOUPA follows a modular design structure. As depicted in Figure 3.12, it is composed of 2 cores: ❶ SOUPA Core or the inner core contains generic ontologies common to all applications and ❷ SOUPA Extension or the outer core contains additional ontologies specific to applications. SOUPA inner Core is composed of base ontologies which can be used and extended to address various application needs. The main core has the following ontologies.

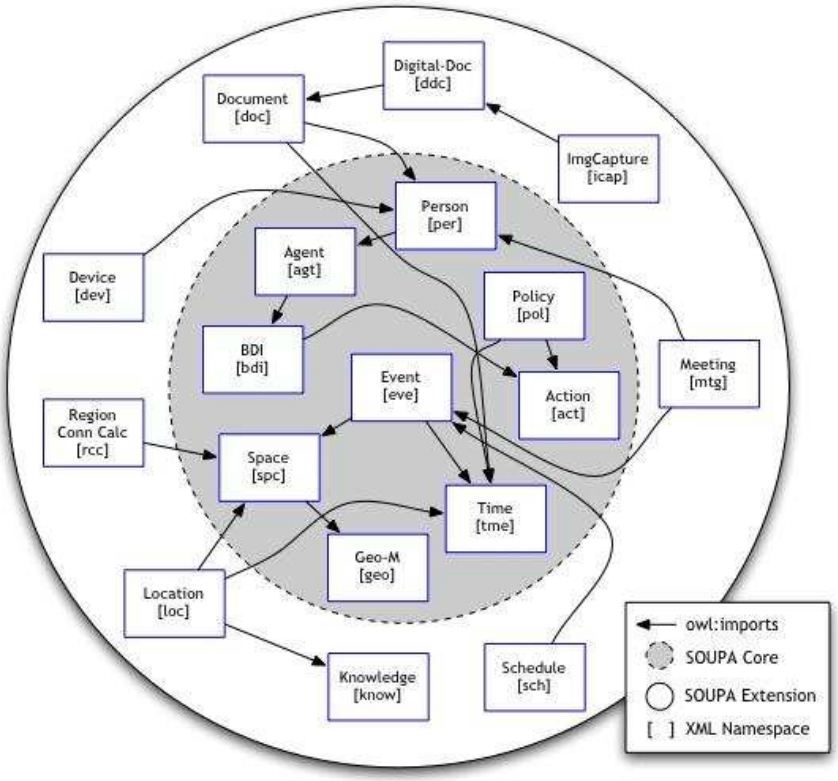

Figure 3.12: SOUPA Ontology

- The *Person* ontology is built around the *Person concept*. In particular, it is used to model the End-User. For instance, the End-User basic profile and enhence it with his preferences and needs, his social and professional information etc.
- The *Agent & BDI* ontologies together are used to model agents interaction with the system. For instance, it is used to model and identify if an agent is a human or a software-based entities. Furthermore, they are also used to explicitly define agents interaction with the system by defining their goals, plans, wants, intentions, convictions and duties.
- The *Policy* ontology provides the vocabulary needed to define security and confidentiality rules. Since ontologies provides a descriptive way to defines rule logic, it makes it possible to deduce reasoning based on theses rules. *Rules* can be defined by the system administrator in order to set the access rights or by the End-User to insure the confidentiality of concerned context information;
- The *Action* ontology describes actions launched by system agents. Each *Action* has one or several associated attributes describing the various actors involved in a particular action. These attributes define what entities will be impacted by a particular action; where and when the action will be executed; what necessary tools will be needed to execute the action; etc. Furthermore, each `Action` can only be executed by an agent if it has the required permission to execute action. These execution rights however, are defined by the rules set up in the *Policy* ontology.
- *Time* and *Entry sub-ontology* that together makeup the Time ontology, enable `SOUPA` to provides a set of ontologies to describe timing and timing-related properties. It relies on `DAML` to enable relationship among timing events by providing *tme:TimeInstant* and *tme:TimeInterval*.
- The *Space* and *Geo-Measurement* ontologies provides the basis for reasoning on spatial relations between geographical areas. The former presents the symbolic representation of a location and related spatial relations, while the latter defines the geo-spatial vocabulary (i.e. longitude, latitude, distance, etc.). These ontologies are used to map geo-spatial coordinates of a geographical location to its symbolic representation and vice-versa. Furthermore, they are also used to represents geographical measurements of a location.
- The *Event* ontology presents events with spatial and temporal extensions. `Events` can be an occurrence of an activity, a result of a forecast or a capture of a predefined trigger.

3.2.4 CONON

`CONON`(CONtext ONtology) [WZGP04] is an ontology designed to representation context information in pervasive environments. As we know, services in pervasive environments address different intelligent spaces (i.e. home, vehicle, etc.), as a consequence, services can generally be groupped as a collection for domains and sub-domains. To address this, `CONON` firstly, aims to represent generic context information suitable to all main domains. And secondly, provide

enough extensibility to represent context information specific to a particular sub-domain. To realize this aim, CONON has been designed as a two-level structure: ❶ high-level ontology, is used to model basic context information and related general properties, and ❷ the lower-level ontology, provides collection of ontologies describing concepts in details, their link to specific domains, and their related properties. Figure 3.13 introduces the high-level ontology, in which the authors choose to model the context around four abstract entities: *CompEnt* (including all software or hardware entities allowing access to the service), *Person*, *Activity* and *Location*. This high-level ontology in CONON represents the Upper part of the ontology which then can be mapped to any domain-specific lower part. Furthermore, to show the use of CONON ontology in a concrete example, figure 3.14 highlights the partial representation of an ontology specific to smart homes. In this representation, the number of abstract classes of the high-level ontology increases with new classes specific to the modeled domain. For instance, the *IndoorSpace* class is extended with four new sub-classes: *Building*, *Room*, *Corridor* and *Entry*. CONON can also

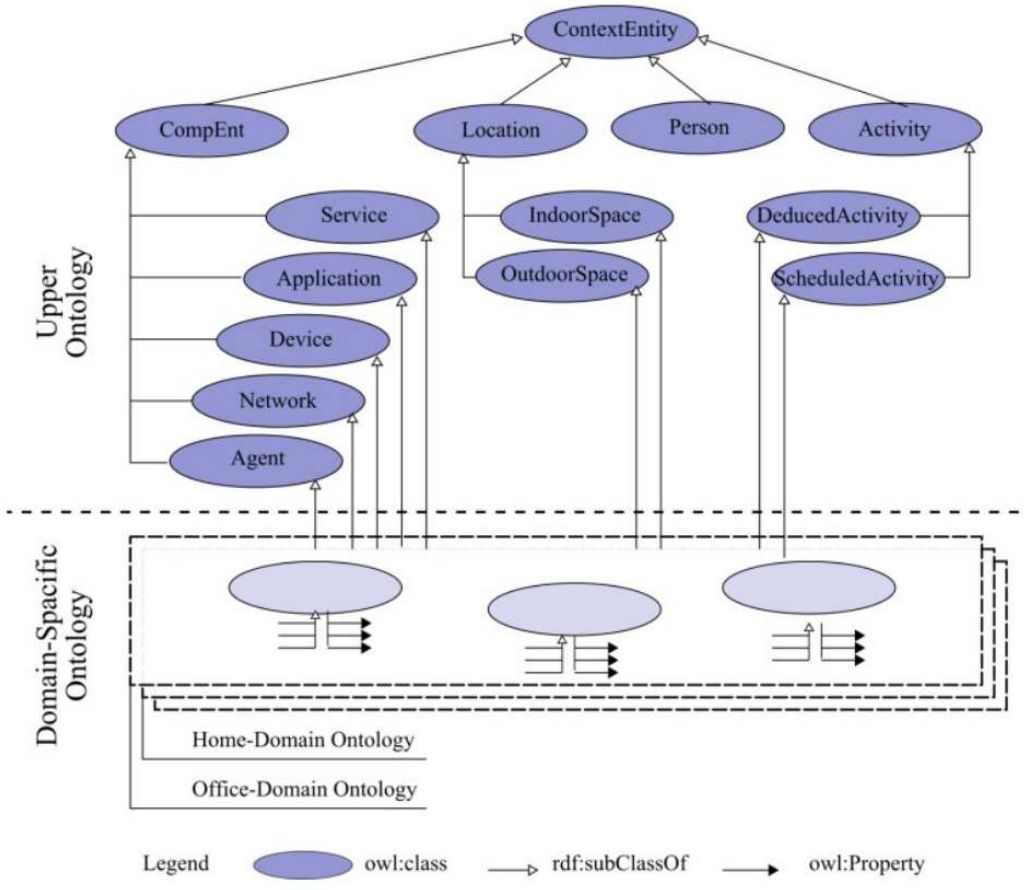

Figure 3.13: CONON Architecture

reason and infer new implicit information from explicit ones. Furthermore, it can detect context inconsistencies due to captured errors. Since CONON is based on OWL the descriptive logic rules

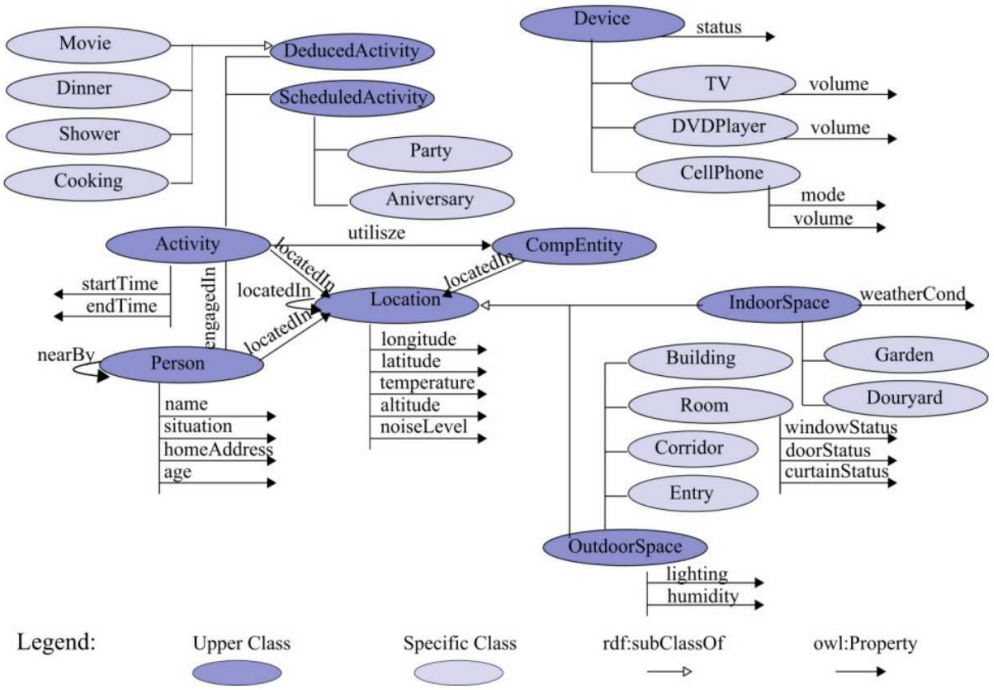

Figure 3.14: CONON Architecture: Smart Home

integrated in OWL semantics makes the reasoning process possible. Moreover, CONON can also be used as a flexible reasoning system to deduce context situations. For instance, with the help of End-User defined rules based on first order logic, various activities of End-User can be deduced from a certain number of facts included in the ontology.

3.2.5 Critical Analysis

In the above section, we reviewed some of the existing ontology-based context models. With the maturity of Semantic Web Languages ontology-based context models have gained a lot of importance. A Context Ontology Language CoOL describes contextual facts and contextual interrelationships by projecting the conceptual base on Aspect-Scale-Context information (ASC) model to language elements. While COBRA-ONT is a collection of ontologies expressed in OWL for context-aware systems. COBRA-ONT defines concepts associated with four distinctive but related themes: places, agents, agents' location and agents' activity. On the other hand, SOUPA, developed by the same authors as COBRA-ONT, deals with pervasive computing and is composed on two parts: ❶ the SOUPA core that holds ontologies which provide a common vocabulary for different pervasive computing environments and ❷ the SOUPA extension which contains ontologies for different domain specific vocabularies. However, to overcome the interoperability issue, SOUPA maps its concepts to some of the well known external ontologies such as FOAF [BM07], DAMLTime [PH04], etc. CONtext ONtology CONON captures the general features of basic contextual entities like (location, user, activity and computational entity) and

uses the domain specific ontology to describe the concepts related to a specific domain. The reported models have shown that ontologies can be perceived as promising tools towards adequate description and representation of context data. Hence, in our solution we plan to utilize and adapt the concept brought forward by CONON which separates generic context information from the one specific to a particular domain. This enable us enhance our solution to provide personalized service discovery by uniquely identifying different entities and their properties.

3.3 Summary

In this Chapter, we did an in-depth analysis of the current state of the art. In order to show a clear distinction, we have separated our analysis into two parts. In the First part, we have reviewed the existing protocol interoperability solutions. Protocol interoperation is current addressed using either *explicit* or *transparent* approach. During our analysis we found that INDISS uses the *transparent* approach but requires a high degree of knowledge about network programming. On the other hand, ReMMoc use the *explicit* approach and provide its own API's while z2z use the generative *transparent* approach and provides a dedicated runtime environment to enable interoperability among discovery protocols. In the second part, we have reviewed the state of the art related to ontology-based context models. During our analysis we have found that ontology-based models support shared understanding of a domain vocabulary, distributed composition and partial validation which makes them a good candidate for modeling context in a dynamic environment. However, we believe that the approach used by CONON is a good candidate for our solution to enable service personalization, adaptation and seamless device mobility. In Chapter 4, we present the first part of our solution; an interoperable service discovery solution for Future Internet called the ZigZag middleware.

Chapter 4

ZigZag Middleware

"The more we share, the more we have "

–Leonard Nimoy

Contents

Discovering a service in a connected network requires in-depth knowledge about the network characteristics, the protocols used and the nature of target devices (laptop, phone, etc.,) that will consume the service. A highly heterogeneous network, as illustrated in Figure 4.1, should allow diverse devices having different discovery protocols to exist and communicate with each other simultaneously. Indeed, there is a strong need for efficient ways of making services accessible to all connected devices in the network. Traditionally, services were provided by dedicated fixed Service Providers. However, from recently, *Prosumers* [Tap96] (End-Users with Service Provision capabilities in addition to Service Consumption ones) have emerged and has completely disturbed the dynamics of services, including the way they are being discovered. Unlike the traditional Service Providers, the *prosumer* is more mobile and has the ability ❶ to publish its services and ❷ to discover available services in its immediate network. Moreover, traditional

Service Providers either used a dedicated SR or follow a P2P approach to indicate service availability. Although a P2P approach is highly scalable, it does not support interoperability, either between P2P protocols or with legacy SDPs. On the other hand, clients supporting only legacy SDPs are not able discovery services advertised by Service Registries if, ❶ both don't belong to the same network and ❷ both don't use the same protocol. A possible solution to cope with these problems is to use a *middleware*. The *middleware* should be able to discover services in highly heterogeneous multi-protocol environment. Furthermore, it should also enable devices supporting legacy protocols to discover, consume and publish their services irrespective of the underlying discovery protocol.

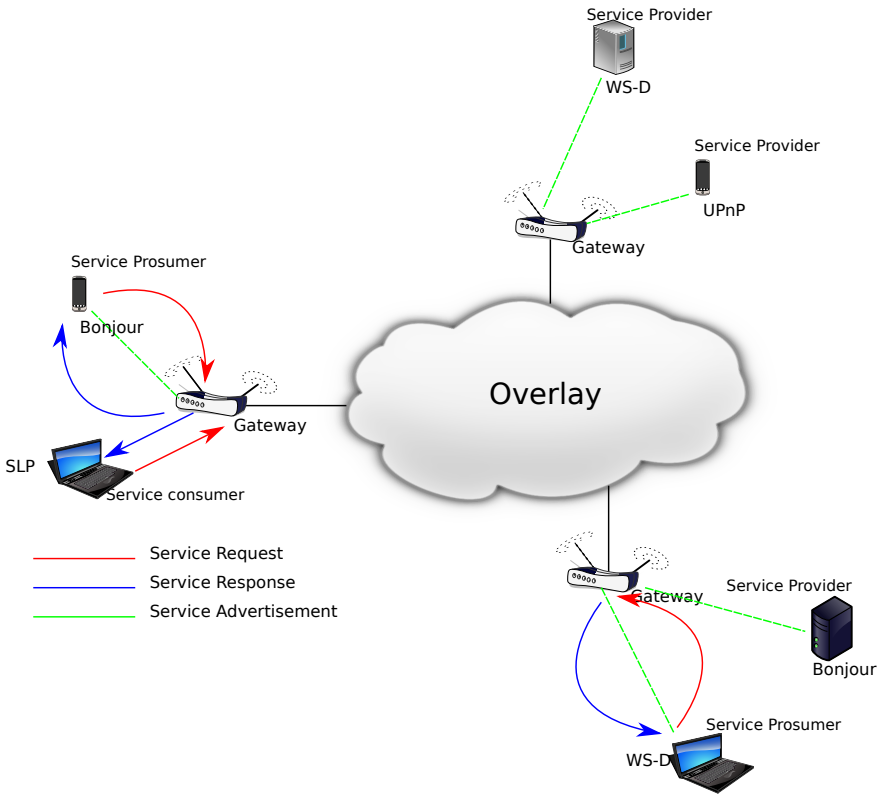

Figure 4.1: Large Scale service Discovery issues

Middleware design requirements

To design a *middleware* that will address the aforementioned issues we have identified four key design requirements.

1. Protocols detection.
   Protocols differ significantly in how they interact with the network. Request may be multicast or unicast, responses may be synchronous or asynchronous and network communication may be managed using a range of transport protocol, most commonly TCP or UDP. A

protocol detection component should be provided to detect all active discovery protocols in the network.

2. Message translation.
   Different protocols provide messages that express information at different granularity. For instance, a single request from one discovery protocol may produce multiple response messages from a different discovery protocol. A message translation component should be provided to address this mismatch among different discovery protocols.

3. Aggregation.
   In a multi-protocol environment a client request may produce multiple responses from multiple providers. An aggregation component should be provided so that a client may receive a single aggregated response for its request.

4. Connectivity among networks.
   To exchange information about available service across network a connectivity component should be provided.

Taking these middleware design requirements into consideration, we have implemented the ZigZag middleware. The next section explains in details the ZigZag middleware architecture.

4.1 Architecture

A key concept common to various Future Internet proposals is to promote an architecture split into two main planes, decoupling services from transport infrastructure. More precisely, one plane is dedicated to upper network layers to provide functions that control and manage service resources for service providers and consumers. The other plane is dedicated to lower network layers to provide functions that control and manage transport resources to carry out data exchanges among service providers and consumers across heterogeneous networks. The split of the architecture enables functions dedicated to services and the ones dedicated to transport to evolve separately and independently. As a consequence, Future Internet offers users unrestricted access to service providers outside their own network boundaries. However, this opportunity raises an issue related to service discovery. Indeed, legacy service providers rely on SDPs that have been initially designed for local area networks. Therefore, making these protocols scale to large scale networks require a substantial effort. In addition, various protocols have been developed to cope with network characteristics and service provider's needs. Thus, enabling service discovery in large scale networks requires managing the heterogeneity of various protocols, deployed in isolated local networks. To this end, we propose the ZigZag middleware.

ZigZag aims to be deployed in isolated local area networks to provide interoperable services discovery in multi-protocol environment. To reach this aim, the architecture of ZigZag has been designed in a modular way to both integrate the state of the art results in service interoperability and service aggregations. ZigZag is architectured around 4 core components, namely: ❶ a

SDP Monitor Component to detect the current SDPs being used, ❷ a *Connectors Management Component* to instantiate the adequate SDP translator, ❸ a *Network Link Component* to maintain connections among ZigZag nodes, and ❹ an *Aggregator Component* to apply aggregation strategies. As depicted in Figure 4.2, these components are plugged together to perform a cross network translation process that is able to translate one SDP to another according to service providers and consumers involvement across heterogeneous networks. The core functionalities of each component are deeply explained below:

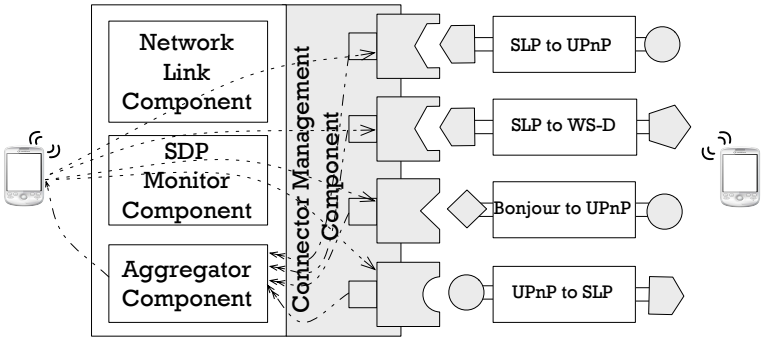

Figure 4.2: ZigZag Middleware Architecture

SDP Monitor Component.
The *SDP monitor* checks the availability of different SDPs in one local environment, as previously introduced by INDISS [BI05]. The SDP monitor is designed to keep track of SDPs currently used. It leverages on the fact that all SDPs use a multicast group address and a UDP/TCP port that must have been assigned by the IANA. Both assigned ports and multicast group addresses are reserved and thus act as a permanent SDP identification tag. The SDP monitor is then able to discover a networked environment by passively listening to the well-known SDP multicast group. More precisely, The SDP monitor learns the SDPs that are currently used from both services' multicast advertisements and clients' multicast service requests. Furthermore, service advertisements are cached locally and are mapped to a UUID to be identified uniquely across different ZigZag nodes.

Connectors Management Component.
A *Connector* translates one SDP to another SDP. It is specific to a pair of SDPs. Thus, there exists as many connectors as different pair of SDPs between which interoperability is required. For instance, in Figure 4.2, four different connectors may be instantiated SLP-to-UPnP, SLP-to-WS-D, Bonjour-to-UPnP, UPnP-to-SLP according to the SDPs being used by either service providers or consumers. A connector is a third party component. Currently, the *Connectors Management Component* supports on the fly instantiation of one or more z2z gateways [BRLM09] that act as connectors. However, ZigZag is not tightly bound to z2z, and may rely on any other translator. Additionally, the Connectors Management Component collects statistics about SDPs being used

to take in charge a fine grained life cycle of instantiated connectors. It may start, stop, pause or resume connectors according to the most often detected SDPs.

Network Link Component.
ZigZag nodes are directly connected to each other irrespectively of the underlying network infrastructure. *Network Link Component* implements a simple protocol for building a data distribution tree among ZigZag nodes enabling them to exchange multicast messages about discovered SDPs, and services across each isolated local area network. The complexity of the Network Link Component implementation depends on the available functions supported by lower network layers. Currently, ZigZag supports a data overlay as an example of network infrastructure for the Future Internet, which provides adequate primitives (join, leave, update, send) to create and/or maintain a logical network among ZigZag nodes. Furthermore, ZigZag can also be deployed on different network infrastructures such as P2P overlay *via* implementation of a dedicated Network Link Components.

Aggregator Component.
The *Aggregator Component* collects a bunch of messages coming back and forth from several connectors instantiated by the Connectors Management Component. More specifically, the Aggregator Component accumulates all SDP responses coming from different remote ZigZag nodes, and selects the one that matches best the criteria of the associated request to then forward it to the service requester. The aggregation logic of the Aggregator Component is described in more details in next section.

4.2 Aggregation

Aggregation is defined as the process of accumulating data from multiple nodes to eliminate redundant transmission and provide value sensitive information to the requesting entity. In broadcast/multicast scenario, the requesting entity has to wait for a fixed amount of time before receiving a response to its request. If the response does not arrive before this time slot the requesting entity signals a timeout and resends the request. However, if the same request is sent to multiple nodes in parallel, response data aggregation is considered as the preferred solution. Response data aggregation ensures that several parallel responses of a particular request are combined into a single response message, thereby reducing the number of messages on the wire and preserving scarce bandwidth. Several papers [KNR02, RAH06, SB07] have shown the importance of aggregation in different network environments. The authors [RV06], presented an extensive survey on data aggregation techniques in the context of sensor networks. In a multi-protocol environment response messages may arrive after the source protocol signals a timeout as different protocols may have different timeouts due to their application design. To illustrate the need of aggregation for service discovery in a multi-protocol environment, we have identified 3 use cases based on different application requirements. In the following configurations, we consider a client $\mathcal{C}$ using SDP $\mathcal{P}_C$, and service providers $\mathcal{SP}_1, \mathcal{SP}_2, \mathcal{SP}_3$ and $\mathcal{SP}_4$ using SDPs $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ and $\mathcal{P}_4$ respectively.

We assume that each service provider belongs to its own local area network and that a ZigZag middleware is deployed on each gateway.

Information craving.
In this scenario, the application running on client $\mathcal{C}$ does not have a stringent timeout constraint. Indeed, the application can wait until all possible responses from available service providers are received. The timeout of the client is greater than each timeout of $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$. Request from $\mathcal{C}$ is sent to all ZigZag nodes. Received responses are then aggregated and a unique response is sent back to $\mathcal{C}$.

Time bound.
In the second scenario, the client application requires replies within a very short period of time. Indeed, the client would signal a timeout if no response is received before its timer expires. Once the request from $\mathcal{C}$ is forwarded to all ZigZag nodes, a timer is started so that an aggregated response can be sent on time, before the expiration of the timer of the client. Therefore, responses that arrive too late are discarded.

Best of both worlds.
The third and last scenario is a mix of the previous ones. In this scenario, the client $\mathcal{C}$ tolerates one timeout expiration and one request re-submission. The request from the client is forwarded to all ZigZag nodes. Responses are cached once they are received and requests are submitted again if one timer expires. Regardless of the responses received, all responses are aggregated and one response is sent back to the client before the second expiration of its timer.

4.3 Evaluation

In order to test the feasibility of our approach we have developed a prototype implementation of ZigZag Middleware. Our current implementation relies on z2z [BRLM09] to dynamically instantiate gateways for protocol translation and an overlay network to connect ZigZag nodes to each others. However, before performing real world experiments using ZigZag, we have setup a simulation to assess how much ZigZag can both reduce the number of messages that flow through the network, and provide value sensitive information to the requesting entity. In the remainder of this section, we explain the simulation parameters and discuss the results we have obtained.

4.3.1 Simulation Setup

To simulate various clients, service providers, network topologies and protocols prior to large scale deployment, we have performed a simulation based on SimPy [Mul12], a network simulator written in Python. To provide the most realistic result, and to outline an accurate evaluation of our prototype, we include in our simulation an adequate model of the *Internet delay space*, which influence inherently the ZigZag performance. In particular, we leverage on a real sample

of the Internet delay space among 3,997 edge networks [ZNN+06] to build our overlay. Correspondingly, we rely on a 3997x3997 delay space matrix that gives all pairs set of static round-trip propagation delays among nodes of our overlay network. Service providers based on either SLP, UPnP, WS-D, or Bonjour are then randomly distributed uniformly over the matrix. Each node hosts only one type of service.

We run our experiments 50 times with three different client instances: C_1 that uses UPnP, C_2 that uses Bonjour, and C_3 that uses SLP randomly located in one node of the overlay. At each run, clients are located in a different node. A request from a client is generated according to the Poisson process with a rate of 5 requests per seconds for a simulation period of 200 seconds. The processing time P_{time} of a service provider, to send responses upon the reception of a request, is $P_{time} = k \times (round\_trip\_delay/2)$ where k is a factor randomly chosen from 2 to 3.6 according to SDPs specification [GPVD99, PFK+08]. An infinite response time means that the service provider is overloaded and that the request has been dropped. The time required by a ZigZag node to translate a message from one protocol to another depends of protocols used by both the client and the targeted service provider. Table 4.1 shows the different possible translation time. To define these time values, we computed the average time consumed by z2z connectors [BRLM09] to perform the translation. In the remaining, we always assume that a ZigZag node is deployed in the local area network of the requesting clients, thus the round-trip propagation delay among requesting clients and its closest ZigZag node corresponds to the round-trip delay of a 100 Mb/s LAN network. The forwarding of SDP requests from clients to one or more adequate remote service providers across the overlay is provided by a service provider selection algorithm that should redirects SDP requests from clients to an appropriate remote service provider, based on factors such as the client location, network conditions, processing load, service search criteria and other parameters dedicated to ALICANTE [ALI13] network substrate. However, such model for ALICANTE is not yet available. Thus, we use a selection algorithm that picks up required service providers randomly in the delay space matrix to get their round trip delay. A more accurate algorithm is planed to be used and will give more efficient simulation results.

	SLP	UPnP	WS-D	Bonjour
SLP	0	0.78	0.84	0.59
UPnP	0.78	0	0.65	0.89
WS-D	0.84	0.65	0	0.90
Bonjour	0.59	0.89	0.90	0

Table 4.1: Average translation time (in seconds)

4.3.2 Simulation Results

The median results of our simulations are shown below. In the simulation results, the x coordinate indicate the number of services that have been reached by clients during the simulation whereas the y coordinate gives the number of generated messages.

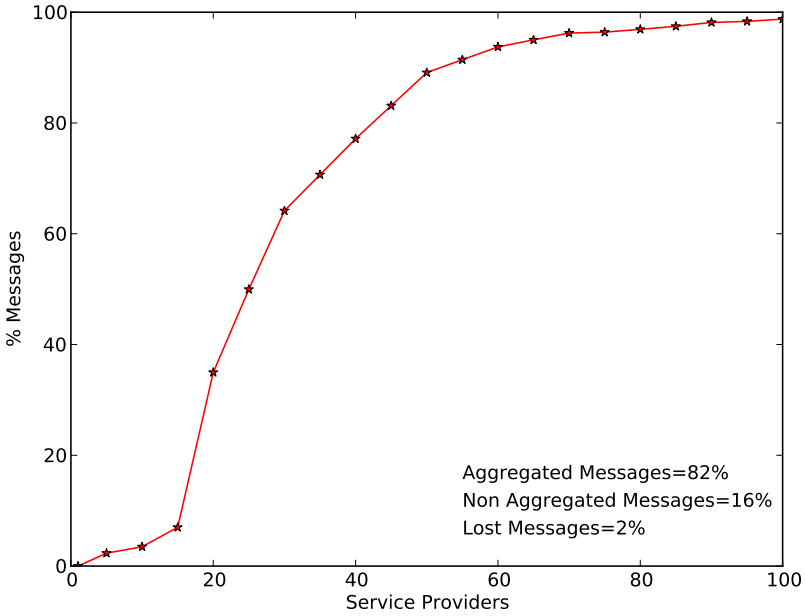

Figure 4.3: Use Case 1: Information craving

Information craving.
Figure 4.3 shows the results of our simulation for the *Information craving* scenario. In this scenario, as the timeout of one client is greater than each timeout of SLP, UPnP and Bonjour, only one client is required to generate requests, and the simulation result corresponds to the best case as the client timeout is high. 82% of the received responses are aggregated, with at least 2 or more responses per message. About 16% of non aggregated messages are received. About 2% of the messages are lost due to errors in the network or replies that did not arrive on time.

Time bound.
Figure 4.4 shows the results of our simulation for the *Time bound* scenario. In this scenario, three different client has been used, one for each SDP, i.e. SLP, UPnP and Bonjour. Each client uses therefore a different timeout. In average, 62% of received responses are aggregated with at least 2 or more responses per message. Further, about 28% of non aggregated messages are received and 10% of the messages are lost due to errors in the network or replies that did not arrive on time. The increase of both non aggregated messages, and lost messages comes from the fact that some replies arrived after the client has signaled a timeout, and are thus ignored. Compared to the previous results, the simulation results obtained in this scenario corresponds to the worst case.

Best of both worlds.
Figure 4.5 shows the results of our simulation for the *Best of both worlds* scenario. In this

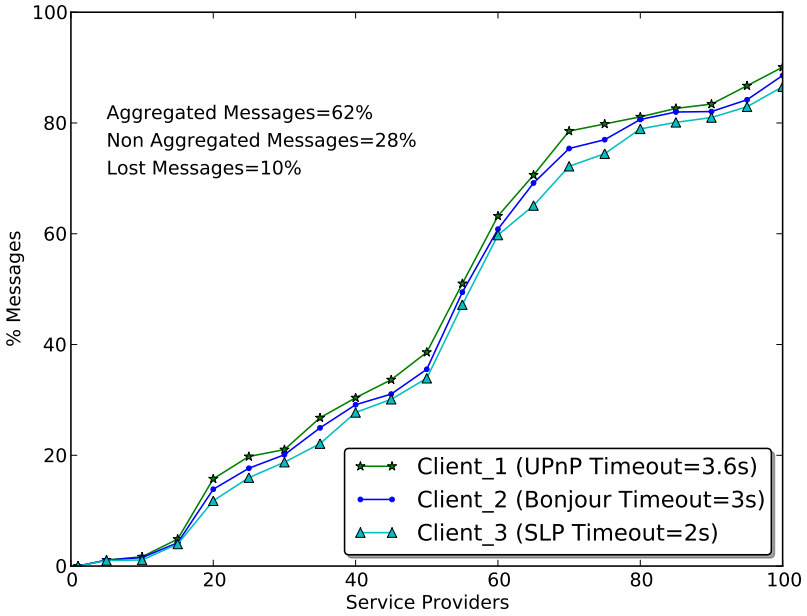

Figure 4.4: Use Case 2: Time bound

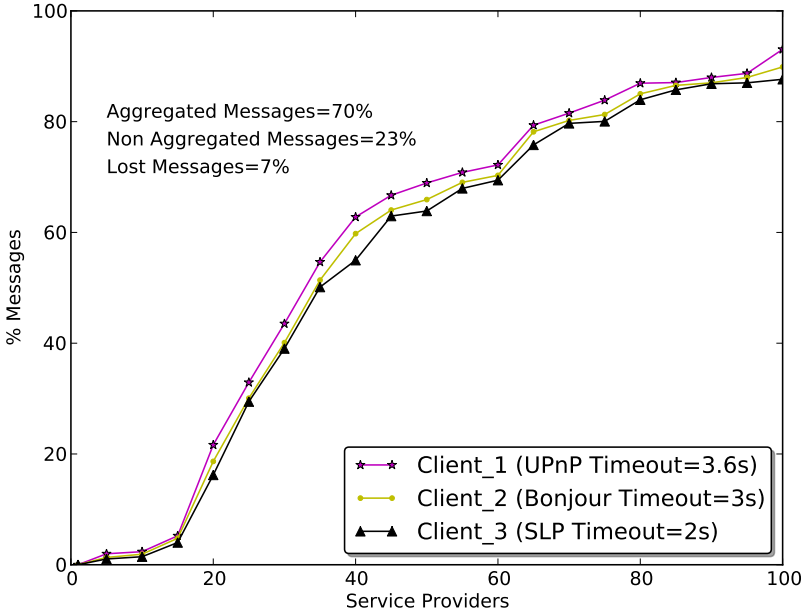

Figure 4.5: Use Case 3: Best of both worlds

scenario, in average, clients get 70% of aggregated responses with at least 2 or more responses per message and about 23% of non aggregated messages. About 7% of messages are lost due to errors in the network or replies that did not arrive on time. The successful increase of aggregated responses comes from the fact that clients tolerates one timeout expiration plus one request re-submission. It means that ZigZag nodes are caching all responses in order to aggregate them to reduce the number of requests sent back to the client.

	Average number of messages received by a client		
	Aggregated	Non-aggregated	Lost/Dropped
Information craving	82 %	16 %	2 %
Time bound	62 %	28 %	10 %
Best of both worlds	70 %	23 %	7 %

Table 4.2: Simulation summary

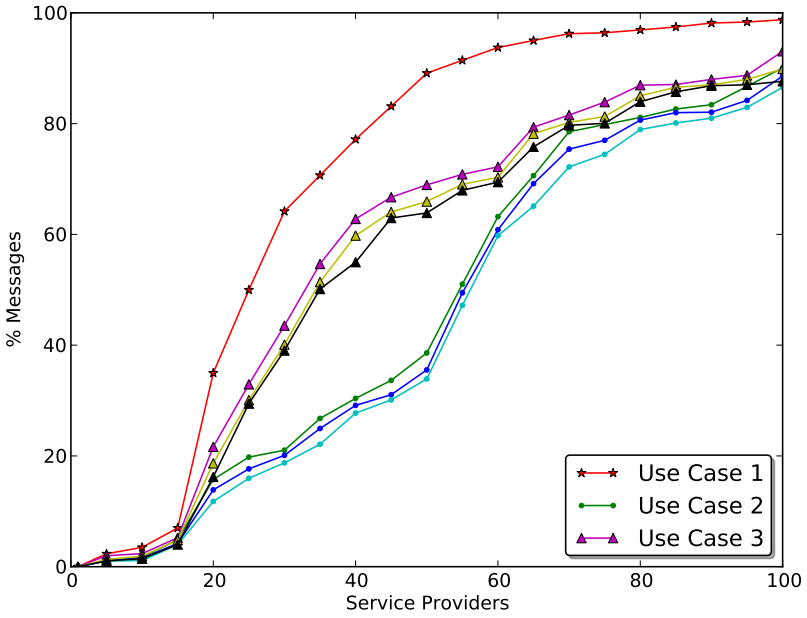

Figure 4.6: Comparison: All 3 use cases

Table 4.2 gives a summary of our simulation results. Our experiments have demonstrated that ZigZag enables 57% to 80% of received messages to be aggregated, having at least two service provider replies. This high rate of aggregated messages implies a significant reduction in the number of messages exchanged through the network and more valuable answers to the client. Figure 4.6 shows a comparison of all the three use cases. In addition, various policies can be

easily implemented and deployed using ZigZag to cope with user expectations and network constraints.

4.4 ZigZag integration in ALICANTE

The ZigZag middleware can be deployed as a standalone system to discovery services in large scale heterogeneous network. However, the modular architecture enables it to be integrated into any existing system. This empowers ZigZag to be customized, to suit once needs. Hence, ALICANTE can utilize this feature of ZigZag in the *Service Environment* to improve its service offering and provide context-aware service discovery. In the *Service Environment* ZigZag can be integrated individually at two critical points, namely; at the HomeBox and the Service Registry.

4.4.1 ZigZag at Service Registry

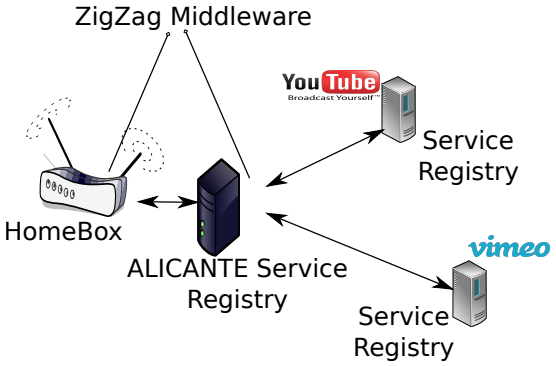

Figure 4.7: ZigZag at Service Registry

The popularity of today's existing social media platforms; (YouTube, Flickr, Facebook, Google+, etc.) permits End-Users to publish their content and consume available services. However, the users need to uses individual platform applications to access available services. Towards this, and as depicted in Figure 4.7 the ALICANTE service registry can utilize the assistance of ZigZag to interact with other external service registries. This permits the EU to use his/her favorite application to access services from any platform. Furthermore, it enables the Service Provider to provision value added services that adhere to users' current context and situation. Moreover, it allows the EU to act as independent content distributor (Media Prosumer) from his/her premises thanks to the always-connected HomeBox device. Figure 4.8 shows the detail integration of ZigZag middleware components in ALICANTE service registry, Furthermore, it also shows the mapping of ALICANTE interfaces with the integrated ZigZag middleware components.

ZigZag Middleware

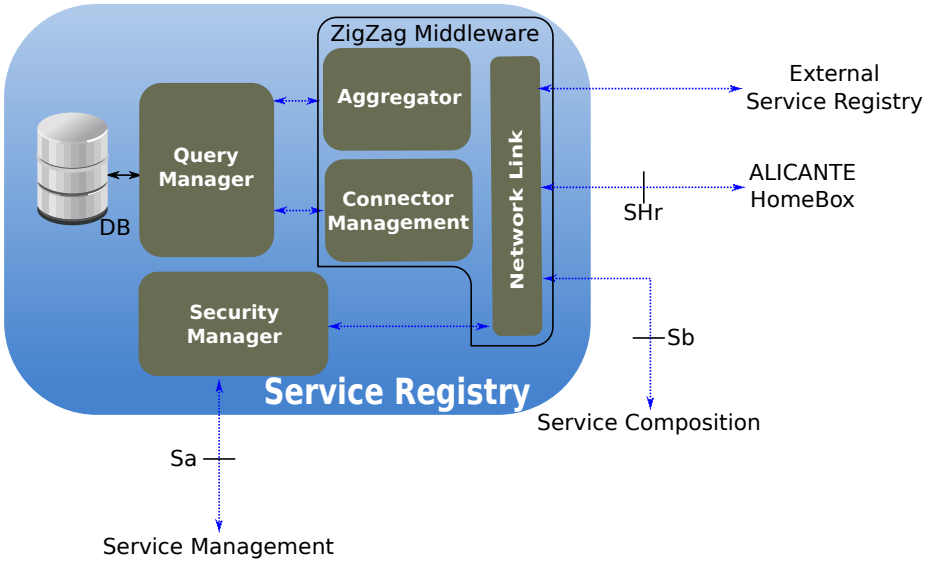

Figure 4.8: ZigZag component integration at Service Registry

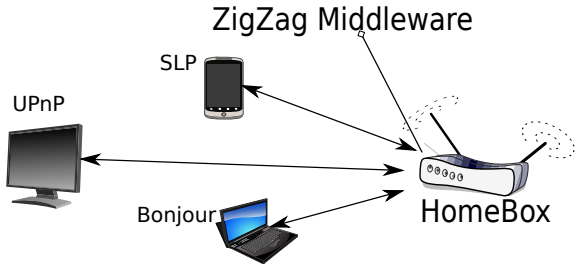

Figure 4.9: ZigZag at HomeBox (simple mode)

4.4.2 ZigZag at HomeBox

At the HomeBox the Zigzag middleware can be configured into two modes, namely, *simple* and *P2P* mode. Figure 4.9 illustrates the typical scenario at a home of a user utilizing the *simple* mode. As shown, the user home is densely populated by different kind of devices such as smart phones, laptops, IPTV and cameras with either wired or wireless interfaces. Each of these devices support different discovery protocols making it difficult for them to interact with each other. When ZigZag is configured in the *simple* mode, the network component of ZigZag middleware uses the same protocol as the ALICANTE SR to search and publish End-User service.

One of the Key features of ALICANTE architecture is a clear segregation of *environments* and *layers*. As mentioned earlier ALICANTE architecture provides two virtual *layers* namely; the Home-box layer and the CAN layer. Leveraging on the virtual Home-Box *layer* ZigZag can be configured in the *P2P* mode. In the *P2P* mode, the network component of ZigZag utilizes the same protocol used by the virtual layer to organize and communicate among HomeBoxes.

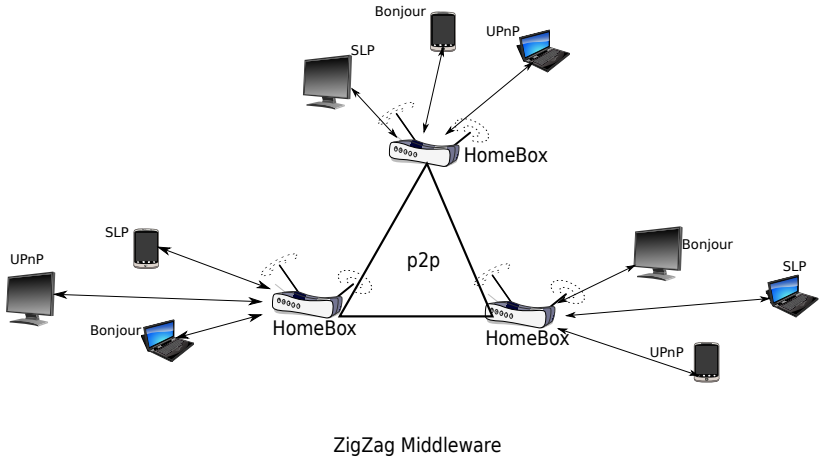

Figure 4.10: ZigZag at HomeBox (P2P mode)

This enables ZigZag to delegate and share resources among connected HomeBoxes. Integrating ZigZag middleware in ALICANTE HomeBox is very advantageous as it manages the heterogeneity of different protocol and enables seamless discovery and access to services in a multi-protocol environment.

4.5 Summary

In this Chapter, we have presented our first contribution the ZigZag middleware. Firstly, we have highlighted the issues for service discovery in large scale network. We have then showed how ZigZag approach can address these issues. Secondly, we have presented the modular ZigZag middleware architecture and the advantages of each of its components. Thirdly, we showed the importance of aggregation in the process of service discovery and evaluated our approach with the help of simulations. Furthermore, we have described the integration of ZigZag in ALICANTE. And finally, we showed the need for a policy language that can help developers defines different policies. In the next chapter we present our policy framework and its features.

Chapter 5

Policy framework for context-aware personalization

"Things should be made as simple as possible - but no simpler"

–Albert Einstein

Contents

The evolution of information technology: centralized, homogeneous systems have been replaced by distributed processors and applications connected by large and widely disparate networks. Furthermore, Web technology is continuously extending the reach of the network, as a results the number of users within the network has increased tremendously. As new technologies are evolving rapidly it has become clear that the existing network and management systems are not very well suited to cope with automated and self-learning requirements in user and application-driven environments. There is therefore a need for application/user aware provisioning and controlling system resources.

The fundamental use of distributed system is to allow connected devices to use shared resources to complete a particular task. However, to use shared resource in a distributed environment at runtime, connected devices need to know if they are able to address the needs of the new incoming requests. To this end, policies have been very successful. A policy enables system administrators to define specific application conditions and associated actions. For instance at runtime, if an application specific condition is matched for a particular incoming request then an associated action or group of actions would be executed by the system. Additionally, to take full advantage of the distributed architecture, connected devices should be able to delegated a job if they cannot satisfy the requirements of new incoming request, hence runtime distribution and managements of policies becomes very important.

Let us take an example to highlight the need for delegation and runtime management of policies in a distributed environment. As depicted (see Figure 5.1), each network is connected to an overlay with the help of a gateway enhanced with ZigZag middleware. The system administrator has configured a default policy on each ZigZag node. This policy forwards any request by a client to the overlay. Furthermore, this policy expects the responses to arrive in the source protocol format and hence blocks enough resources to aggregate them when they arrive. Additionally, each

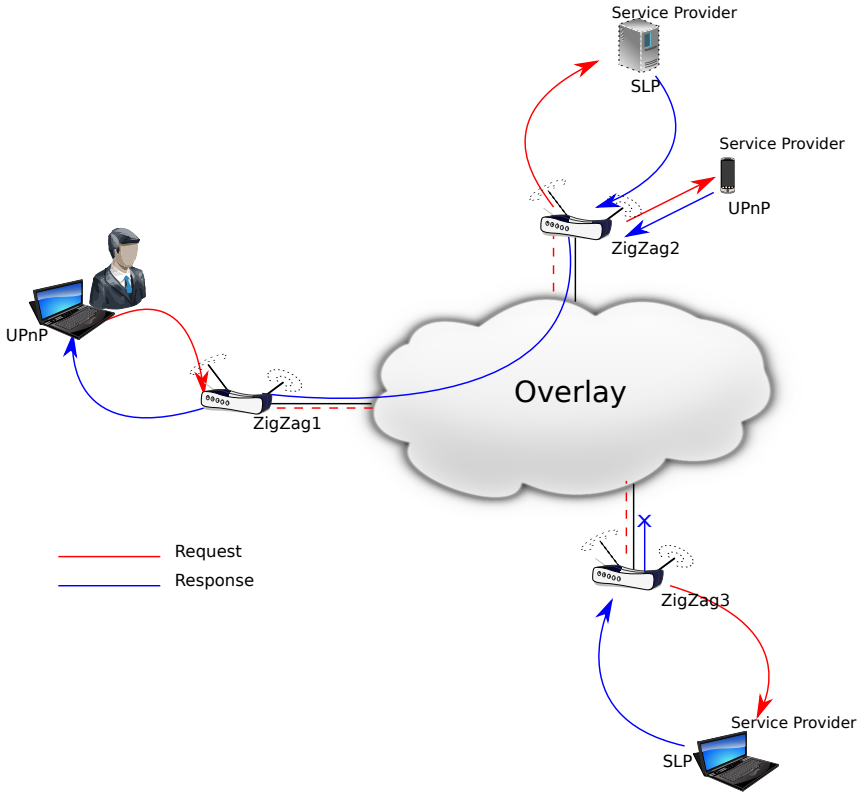

Figure 5.1: ZigZag middleware: need for delegation

ZigZag node only supports translation of messages from one protocol to another for the SDPs

present in its local environment. In the above example when a client device (laptop) supporting UPnP protocol makes a search request to ZigZag1, it forwards this request to the overlay on behalf of the client. This forwarded request is received by all active ZigZag nodes connected to the overlay. Further, when the request is received, each ZigZag node extracts the request parameters and matches them against the policy conditions defined earlier by the system administrator. For the above request, although ZigZag3 has a service match, it cannot send a response, as the necessary translation mechanism is absent on ZigZag3. However, if ZigZag3 was able to delegate this task (translation) to ZigZag2 then it would have been able to reply to ZigZag1. Furthermore, if ZigZag3 were able to configure policies on ZigZag2, it would have significantly reduced the number of messages in the network. We propose to address the issues faced by ZigZag3 in the above example with the help of a policy framework.

Policy Framework design requirements
To design a policy framework that address the aforementioned issues we have identified four key design requirements.

1. Delegation
   The main idea behind the use of delegation is to get a job or task done by some other entity. However, to complete the task the entity should have the capability and the resources to complete the task at hand. To achieve this, a delegation mechanism should be provided to enable entities to request resources from other connected entities.

2. Dynamic configuration and logging.
   To be able to direct system flow at runtime, a dynamic policy management mechanism should be provided. This mechanism should be able configuration policies at runtime. Furthermore, a logging mechanism should also be provided to log system states.

3. Facilities for domain specific customization.
   In a highly distributed environment messages are strongly domain dependent. To address this, a customizable messaging paradigm should be supported that can handle both request-response and request-only communication.

4. Context-aware policy condition.
   In a highly pervasive environment consisting of different elements, it is very difficult to uniquely identify different elements and their properties. To address this, a policy generation mechanism should be provided that can generate context-aware policy conditions.

Taking these design requirements into consideration, we have implemented a context-aware policy framework. The next section explains in details our policy framework architecture and its advantages.

5.1 Policy Framework

To enable system administrator to configure policies on ZigZag nodes, we have implemented a context-aware policy framework. Our policy framework relies on the Internet Engineering Task

Force (IETF) functional architecture. The IETF [YPG00] and Distributed Management Task Force (DMTF) working groups, work together in the creation of different standards related to policies. The IETF is in charge of the architecture definition and the DMTF defines a standard information model. However, the IETF do not use a language for specifying policies, but represents policies with an object-oriented information model. This model is an extension of the Common Information Model (CIM) that is developed by the DMTF. The IETF policy working group defines policies as *"an aggregation of rules, where every rule includes a set of conditions and its correspondent set of actions"*. As depicted in Figure 5.2 the IETF policy framework consists of the following components, namely; ❶ the policy management tool, ❷ the policy repository, ❸ the policy consumer and ❹ the policy target. The functionality of each of these components is explained below.

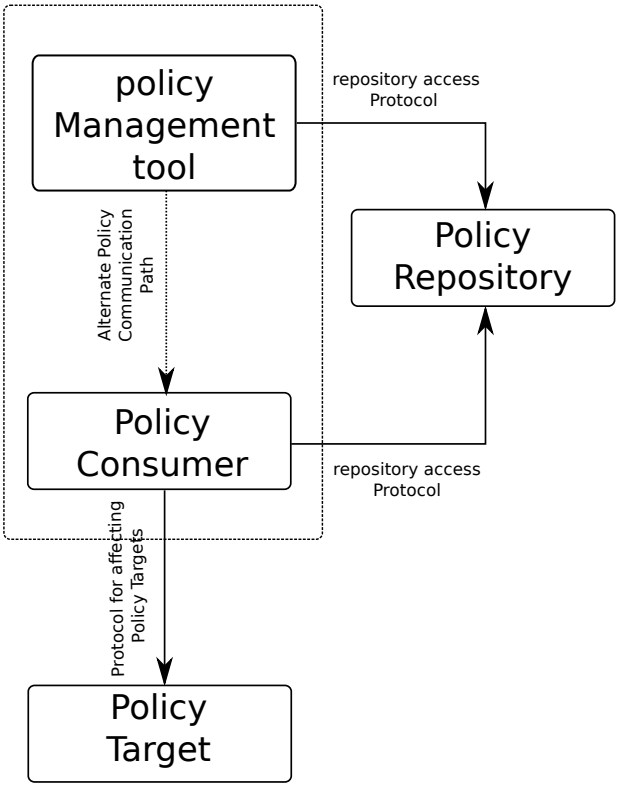

Figure 5.2: IETF Architecture

Policy Management

The Policy Management Tool produces POLICY RULES, which are used by the POLICY CONSUMERS to appropriately influence the behavior of the POLICY TARGETS. The POLICY RULES specify the logic used to deliver the prescribed service and service-levels. The POLICY CONSUMERS on the other hand interpret and may further validate the POLICY RULES and then map these rules to the underlying policy mechanisms of the POLICY TARGETS. Furthermore, the

POLICY CONSUMERS may also transform POLICY RULES into forms that POLICY TARGETS can use directly.

Policy Repository
In a policy repository, POLICY RULES are represented as a set of object entries that include object classes for POLICY RULES, policy conditions and policy actions. These POLICY RULES co-reside with objects representing network devices and services as well as with other objects representing entities like users, printers, and file servers.

Policy Consumer
The POLICY CONSUMER is responsible for POLICY RULE interpretation and initiating deployment. Its responsibilities include trigger detection and handling, rule location and applicability analysis, network and resource-specific rule validation and device adaptation functions. In certain cases, it transforms and/or passes the POLICY RULE and data into a form and syntax that the POLICY TARGET can accept, leaving the implementation of the POLICY RULE to the POLICY TARGETS.

Policy Target
POLICY TARGETS are responsible for the evaluation of policy conditions. Furthermore, it also handles the execution of actions. Moreover, it can also perform related device-specific functions, such as POLICY RULE validation and policy conflict detection.

As mentioned before, our policy framework relies on the IETF functional architecture. However, we have enhanced the functionality with the addition of a policy language. Our policy language enables system administrator to write and configure policies on ZigZag nodes in a clean and effective way. As depicted in Figure 5.3 the Policy Framework is made up of 3 main components, namely; the `Policy Engine`, the `Policy Language`, and the `Policy Generator`. The functionality of each component is explained below.

5.2 Policy Engine

The `Policy Engine` consists of four modules namely; ❶ *Policy Parser*, ❷ *Policy Evaluator*, ❸ *Policy Manager* and ❹ *Policy Action*. The functionality supported by the modules are briefly explained below.

Policy Parser
As it turns out typos and errors are bound to creep into a policy during its generation. In order to avoid system crush at runtime, syntax checking could go along ways before the policy can be used in the system. To this end, our Policy parser adheres to the context-free language described in APPENDIX II and makes sure that the policy instance is validated before it is utilized in the system. Furthermore, the policy parser also checks and validates the policy actions and their

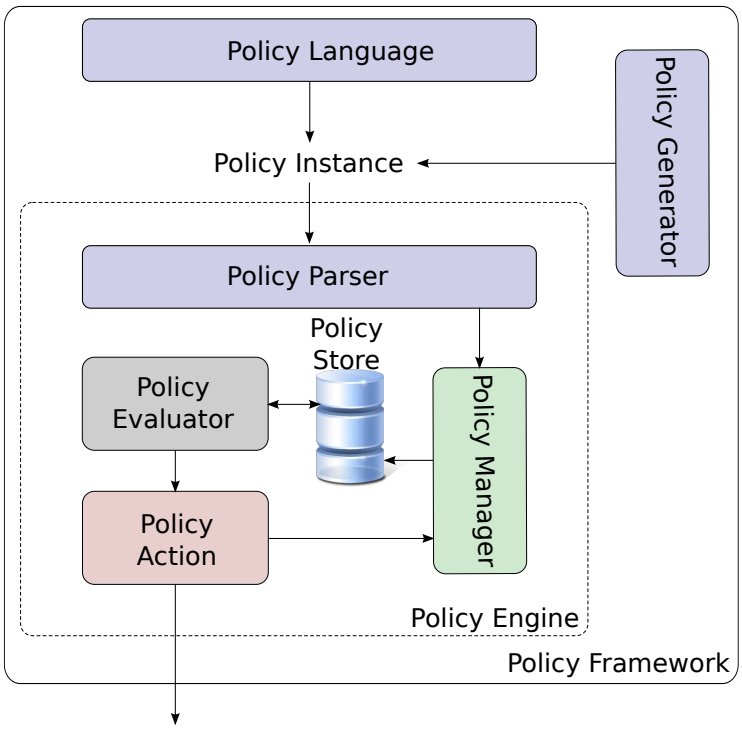

Figure 5.3: Policy Framework Architecture

parameters before they are stored into the *policy store*. Hence, any policy instance that does not pass the policy validation check is directly rejected. While on the other hand any policy instance that pass the policy validation check uses the functionality of the policy manager so that, it can be stored in the *policy store*.

Policy evaluator

The policy evaluator has two main responsibilities. Firstly, it is responsible to validate the result of an incoming request with the stored policies. And secondly, it is responsible to direct system flow. To achieve the first objective, the policy evaluator uses the concept called boolean matching. Using boolean matching each condition in the policy is validated with the keywords and values from the incoming request; the result is stored as either true or false. This process is repeated for all the policies in the policy store until a policy match is found (i.e. the final boolean result of all conditions evaluates to true). In case of non valid policy match, the user request is dropped. However, the system administrator can configure a default action for all non valid matches or log the user request. While on the other hand on a valid policy match the *Policy Evaluator* passes the control to the *Policy Action* module.

Policy Manager

Distributed systems management involves monitoring the activity of a system, making management decisions and performing control actions to modify the behavior of the system. However,

the sheer size and complexity of large distributed systems has resulted in a trend towards automating many aspects of management, chief among them being the ability to self-manage policy distribution. Thus there is a growing need to enable policy managers to specify, represent and manipulate policy information to enable self-management in a dynamic distributed environment. Towards this, our policy manager supports the following `policyOperation` functions:

- `add` – Used to add a policy to the *policy store*.
- `remove` – Used to remove a policy from the *policy store*.
- `enable` – Used to enable an existing policy from the *policy store*.
- `disable` – Used to disable an existing policy from the *policy store*.
- `move` – Used to change location of a policy in the *policy store*.

These functions enable the policy manager to self-manage policy entries on a remote or a local policy store with the help of management functions. These functions are described in details in the next section.

Policy Action
Policies encapsulate a representation of information affecting component behavior with the help of policy action. Thus it is highly advantages to specify the scope of a policy action to a particular domain. This enable the policy manager to identify what policy apply to a domain and then use this information to modify the component behavior to achieve system goals. To this end, the policy actions module stores domains component information as action objects. Furthermore, these actions objects are then used to consult with the policy manager before the execution of a policy action. Section 5.3.3 clearly shows the advantages of the system defined policy actions. Further, these policy actions are designed to be domain independent and can be easily configured to cater to different domain needs.

5.3 Policy Language

Our policy language follows the IETF recommendation of `Condition-Action` paradigm. The policies have the shape of: *"If there is (a set of conditions), then (a set of actions) can be done"*. The Following section explains the syntax, constructs and usage of our policy language.

5.3.1 Policy Syntax

The BNF-like syntax of our policy language is shown below. `typewriter font` is used for keywords. *italic font* is used for nonterminals. Parentheses are used for grouping. A superscript $*$ denotes 0 or more occurrences of the preceding item. A superscript $+$ denotes 1 or more occurrences of the preceding item. A superscript $?$ indicates that the preceding item is optional. Comments, begin with # and only after the actions.

policy_spec ::= `policy` *policy_condition* `then do action`
::= *policy_action* (`comment`)$^?$
policy_condition ::= *conditions*
policy_action ::= *functions*
comment ::= `# comment or short policy desc`

Our policy language relies on two basic constructs namely; *policy_condition* and *policy_action*. Using these constructs our policy language is able to influence the runtime behavior of a system. The *policy_condition* is used to validate different runtime requisites, while the *policy_action* is used to direct system flow. The following section describes how we can represent a *policy_condition*.

5.3.2 Policy Condition

A *policy_condition* provides the means to validate the occurrence of a case at runtime. However, to identify and validate different cases with subtle variations, *policy_condition* must support rich set of operators. Furthermore, these operators should be able to accurately distinguish one case from the other. Moreover, a policy should be able to combine multiple expressions and evaluate it as a single *policy_condition*. Towards this, our policy language supports two types of *policy_condition* namely; *Simple* and *Complex*.

policy_condition ::= *conditions*
conditions ::= *Simple* | *Complex*

Simple
A *Simple* condition has the following form:

Simple ::= *iden oper val*
iden ::= *letter* | (*letter*)*
oper ::= < | != | = | > | <= | >=
val ::= *digits* | *letter*

As described above, our language supports the following operator: =, !=, <, <=, >=, > . These operators can be used to indicate subtle differences and uniqueness of each policy. This in turn helps to minimize policy conflicts at runtime. Furthermore, the *iden* in the expression is dynamically resolved at runtime; depending on the *oper* the value of the right operand *val* is stored in the left operand *iden*. This enables us to use the value of the *iden* for a different expression in the same *policy_condition*.

eg1. userID = John
eg2. Location = office

The above examples show how we can use the operator (*oper*) to define *Simple* condition. In the example, *iden* userID is assigned the value `John`. However, if we want to combine the two *Simple* conditions we would need a different set of operators. To achieve this, we propose the *complex* condition.

Complex
A *Complex* condition enables us to combine multiple *Simple* or *complex* conditions to build a more comprehensive set of *policy_condition*. To this end, our language supports the following operators: `AND, OR, NOT`. In addition, we also support the `IN` operator which is used in conjunction with `collection` keyword. Therefore, a condition specified with a `IN` operator, is said to be true only when the condition is part of a `collection` of conditions.

Complex ::= *Simple op Simple* | `NOT` *Complex* |
::= *iden* `IN Collection {`*Complex*`}`
op ::= `AND` | `OR`

e.g., userID = John `AND` userID `IN collection{`list_of_users`}`

The above condition checks if a user with userID *John* is among the users specified in *list_of_users*. The *list_of_users* parameter can be a static comma separated list or a dynamic list generated using a call to an external program. This ability of our policy language to generate a dynamic list at runtime enables the specified collection to be up to date at access time.

5.3.3 Policy Action

A *policy_action* is a set of instructions that specifies parameters used to provide a different quality in one or more services. If the set of *policy_condition* is true, then the *policy_action* or a set of correspondent *policy_action* is executed. *policy_action* dynamically control the system flow at runtime by executing the defined set of actions. In order to provide a self-managing framework, our Policy Language supports the following actions types, namely; *Delegate, Configure, Logging*. These actions are domain independent and the system administrator has the flexibility to customize them to adhere to a specific domain. However, if a system administrator wants to

add different functionality or wants to automate a commonly used functionality he can still do so with the help of standard communication methods called *MEP*. *MEP's* are communication methods that can be customized to accommodate domain specific needs.

policy_action ::= *functions*
functions ::= *function* | (*functions*)$^{?}$
function ::= *Delegate* | *Configure* | *logging* | *MEP*

Delegate
The centralized, labor-intensive management paradigm has been stretched to its limits by emerging large scale, complex, multi-domain networks. There is a growing need for new technologies that can automate and distribute management functions to accomplish scalable and robust operations. Delegation is once such useful management function that support a new paradigm for automated distributed management of networked systems. A *Delegate* Action enables nodes in highly distributed network to depute work to other connected nodes. The *delegate* action is represented as follows:

Delegate ::= `delegate (` *delegateJob*`,` *delegateMessage*`,` *address* `)`

where, *delegateJob* is the identity/name of the job a node needs to assign to another node. The *delegateMessage* is message that accompanies the job name stating precisely the nature of work that needs to be done. To support domain independence the *delegateMessage* uses XML message format. Lastly, the *address* is responsible for sending the request to a destined location or to a network overlay. On execution of *Delegate* action, an unique *returnID* is returned that is used by the policy manager to update the *policy store* on a specific node.

Configure
We have just seen the importance of *delegate* action as a management function. However, in order to use the resource identified after the execution of the *delegate* action we need to update the policies in the policy store so that other actions can use it to accomplish their task. Towards this, we have the *Configure* action. The *Configure* action is used to update policy store entries on one particular node or all the nodes in the network as required by the system administrator. This functionality ensures that the policies in the network are dynamically managed without restarting the system. The *Configure* action is represented as follows:

Configure ::= `config (` *policyOperation*`,` *policyInstance*`,` *address*`,` *returnID* `)`

where, *policyOperation* defines the operations a system administrator can execute on a policy store. These operations are controlled and the execution is authorized by the *policy manager*. All the support *policyOperation* functions will be explained later in the section. Further, the *policyInstance* is an instance of a particular policy that needs to be updated or added to the store. The *returnID* is an UUID returned after the execution of the *Delegate* action. This UUID indicates the that the node has the required authorization to execute the *Configure* action on the remote node. However, if the policy needs to set on all the remote nodes then the system administrator uses a specific key as returnID and the policy is said to a global policy. On the other hand a *returnID* with value `None` is used to execute operations on the local node itself.

Logging

One of the most important tools used to monitor system states in computing is by a process called *logging*. This has become the de facto standard in system auditing. Furthermore, as all the process in the system use messages it is also used for intrusion detections. The *logging* feature provides information like *"who, what, when, where, and how"* thereby enabling the system administrator to make informed business decisions. To this end, we provide the *Logging* action. The *Logging* action is represented as follow:

logging `::=` `log (` *logMessage* `)`

where, `logMessage` is the message that provides additional textual information about the actions or the current process.

Message Exchange Patterns (MEP)

In a highly distributed environment messages are strongly domain dependent. Each domain has its own message interaction mechanism, some support request-response paradigm while other support request-only paradigm. To cater to these domain specific messaging paradigm we support two standard *MEP*. The two *MEP* are represented as follows:

MEP	`::=`	*request-response* `|` *request-only*
request-response	`::=`	`transmit_rr (` *parameters* `)`
request-only	`::=`	`transmit_ro (` *parameters* `)`
parameters	`::=`	*actionType*`,` *messageBody*`,` *address*`,` *messageID*

where, `transmit_rr` is used for request/response message interactions. While `transmit_ro` is used for request-only message interaction. *actionType* is used to uniquely recognize a domain specific action. *messageBody* provides additional information about the action. To ensure compatibility with different domains the *messageBody* support XML message format. While the *messageID* is an UUID used to uniquely identify a messages in the system.

5.4 Policy Generator

Policies can help us to dynamically control system flow and reduce the cost and time generally associated with coding of static systems. However, it is extremely important that the generated policy fully reflects the advantages of the model from which it is produced. In a highly pervasive environment consisting of different elements, it is very difficult to uniquely identify different elements and their properties. Towards this, the semantic community showed some promising results with the help of ontologies. Ontology provides a formal description and semantic for context information in term of objects, concepts, properties and relations. Therefore ontologies have become the most widely used tool for modeling context information in pervasive computing domain. Hence, to take advantage of the feature proposed by ontologies we choose OWL [MVH+04] to model our domain.

OWL has been successfully used by developer for applications needing a classification hierarchy, simple constrain feature and maximum expressiveness without losing any computational completeness. As depicted in Figure 5.4, in our model we identified 4 interrelated entities that are generic to any domain: namely; ❶ the user, ❷ device, ❸ network, and ❹ service. These entities are modeled as `owl:Class` elements and the relations as `owl:ObjectProperty`. Each `owl:Class` is characterized by different `owl:DataProperty` that are considered relevant to the domain.
The `User` entity is described in different profiles:

- The `GeneralProfile` contains general information about the user such as name, age, etc..
- The `SubscriptionProfile` contains information on the different services for which the user have subscribed and the services that he may access.
- The `ContactProfile` contains the contact information of the user such as his address, phone number, SIP URI, etc.
- The `Affiliationprofile` contains information about the different organization to which the user is affiliated.
- The `AuthenticationProfile` contains information that allows the user to be authenticated.
- The `PreferenceProfile` contains the user-defined preferences or the deduced preferences from usage. The user preferences could be generic and applied to any service or situation or they could target a specific service or context entity and thus be applied only when the latter is involved.

The `Device` entity is described as two sub-entities namely: ❶ `HardwarePlatform` and ❷ `Softwareplatform`. The `HardwarePlatform` entity is modeled in a hierarchical way since the components can be either atomic or composite. On the other hand the

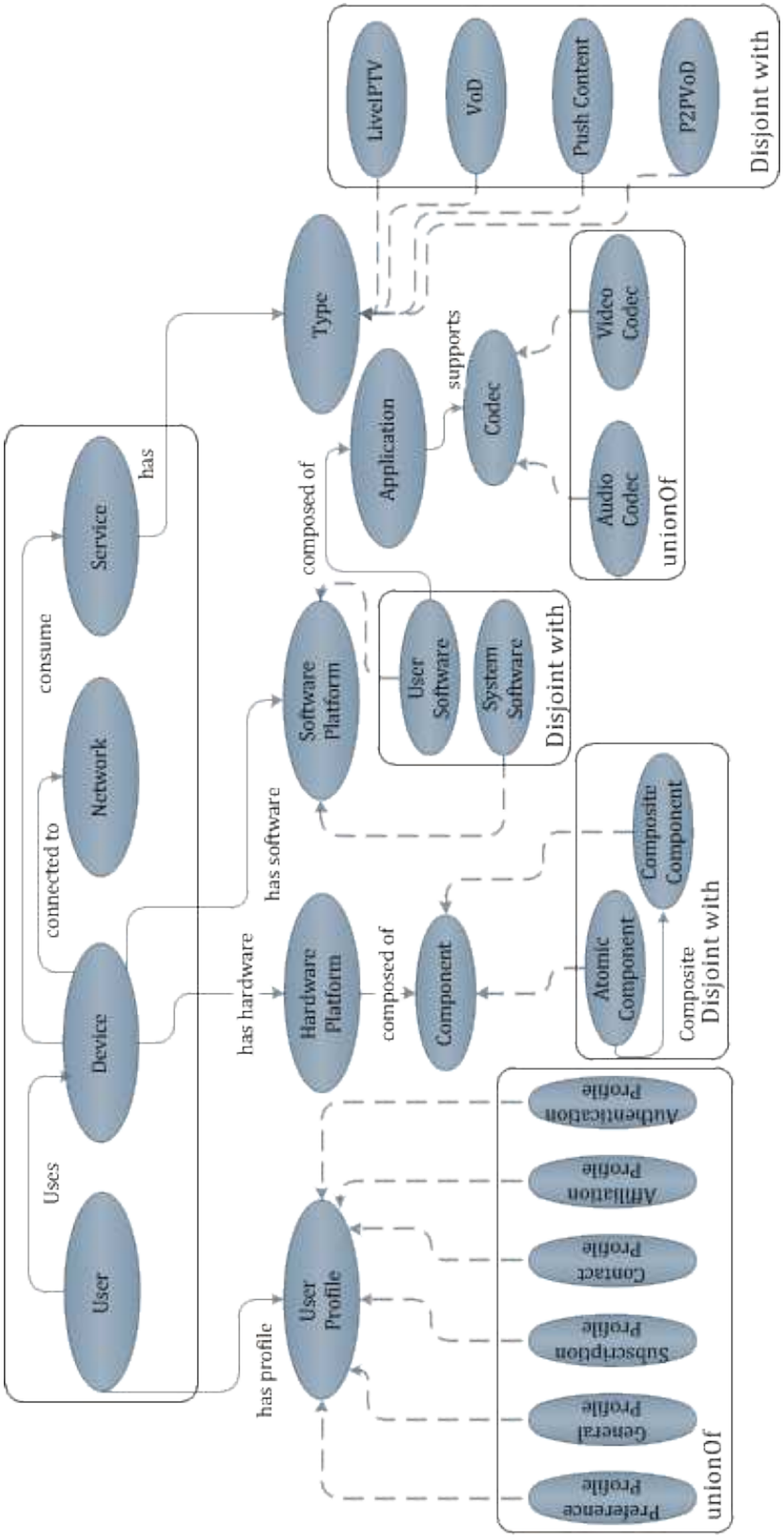

Figure 5.4: Context model

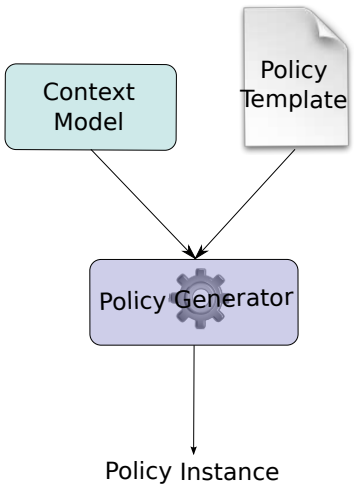

Figure 5.5: Policy Generator

`Softwareplatform` entity represents User and System software's that runs on the device. For instance, audio/video codec or players. The description of the `Network` entity comprises of information such as the name of the network and the theoretical parameters that characterize it. For instance, user location or a loss or error rate experienced in a multimedia session within a related network. These parameters are either reported by monitoring modules or evaluated using subjective techniques. The `Service` entity represents the different services that the user can access. Any entity that provides a service is represented within this entity. The service entity is modeled using OWL-S [MBH+04] Service Profile that relies on Inputs, Outputs, Preconditions and Effects (IOPEs) as modeling parameters.

The use of ontologies helped solve the issue of uniquely identifying the elements and their capabilities. However, in order to effectively use them the policy generator has to present them in the form understood by the system. Towards this, and as illustrated in Figure 5.5 the generator use the service of a template. Templates provides the mandatory policy building blocks leaving enough gaps to be filled by using the dynamic information extracted from Ontology based modeled data.

5.5 Towards better QoE

To provide better quality of service for users, multimedia service and network providers endeavor to improve, the presentation quality of multimedia content and at the same time optimize the network infrastructure. The following example (see Figure 5.6) highlights the use of a policy language towards this objective. The network and service providers can use the policy language to describing policies for a particular user or a group of users based on his/her preference. For instance, the above example shows how the policy language is used to selects the category and video resolution according to the user preferences, location and device.

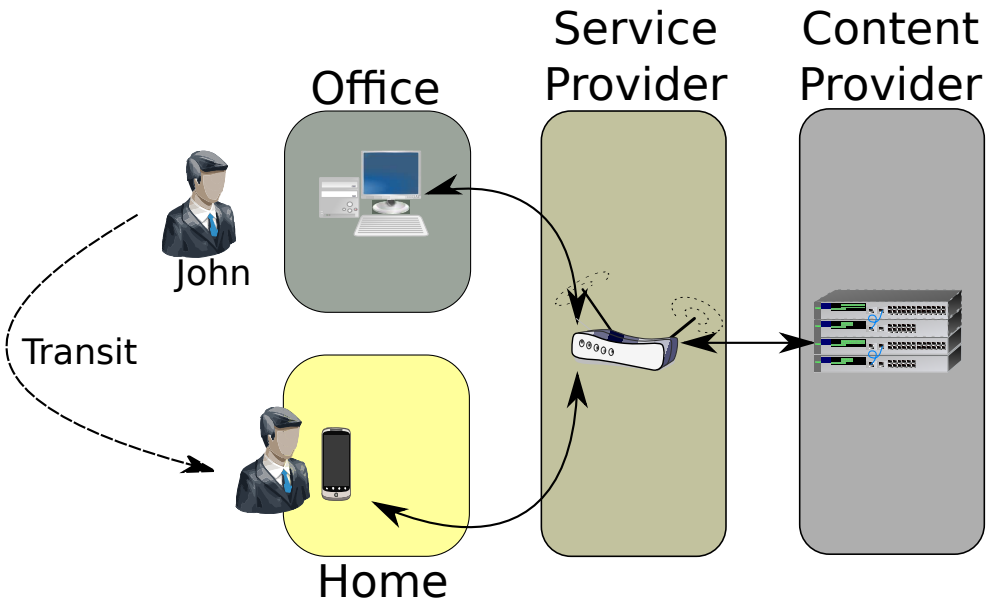

Figure 5.6: Policy Usage Example

```
policy
      (Bandwidth > 256kbps) and
      (UserID=John) and
      (ServiceType=VOD) and
      (Location=fixed) and
      (DeviceType=laptop) and
      (ConnectionInterface=wired)
then do action
      transmit_rr(forward, HD resolution, laptop address,
                  msg-id=123)
      log("Request sent for adaptation, msg-id=123")
```

The policy states that when user JOHN is at his office premises, he prefers to get the match highlights of recently played football matches during his lunch time, when he requests for a VoD service. As JOHN'S office has a very good network bandwidth the policy selects the High Definition (HD) resolution video for JOHN. The simple policy that is generated using JOHN'S current context information for HD content streaming is shown below. However, user JOHN now wants to continue viewing the same VoD service that he was watching at lunch time from his mobile device on his way home. He now logs in from his mobile phone and accesses the same VoD service. At this point, the network entity detects that JOHN'S bandwidth is not enough to consume HD content and hence selects the policy to stream SD resolution optimized for mobile viewing. Furthermore, the video starts to stream from the same position that JOHN paused on the office computer. The updated policy is shown below.

```
policy
      (Bandwidth < 256kbps) and
      (UserID=John) and
      (ServiceType=VOD) and
      (Location=mobile) and
      (DeviceType=phone)
then do action
      transmit_rr(adaptation, SD resolution, adaptation.py,
                  msg-id=123)
      log("Request sent for adaptation, msg-id=123")
      transmit_rr(forward, adapted data, VideoStart=03:12,
                  phone address, msg-id=123)
```

5.6 Summary

In this Chapter, we have presented our second contribution, the Policy Language. Firstly, we have described the need for policies in ZigZag middleware. We have then used the recommendation proposed by IETF and DMTF working group to design the policy framework. Furthermore, we have enhanced the IETF recommendation with the help of a policy language. Moreover, we have proposed a policy generator that relies on ontologies to generator context-aware policy conditions. And finally, we have described how policies can be used to enable better QoE for the End User.

Chapter 6

Conclusion

"Success is a journey, not a destination, doing is often important then the outcome"

–Arthue Ashe

Contents

Due to rapid innovations in modern information and communication technologies, there has been a substantial increase in users, devices and services connected to the Internet. However, in spite of the current Internet's overwhelming success, there are growing concerns about its future and its robustness, manageability, and scalability. Future Internet is presently seen as a large scale infrastructure that can provide dependable management, scalability and robustness among heterogeneous networks. Furthermore, users in Future Internet should be able to use *anytime*, *anywhere*, remote services on *any device* regardless of their underlying technology. Currently, a vast bulk of services primarily connected to the Internet has not been designed to interact seamlessly with each other. These services rely on contrasting Service Discovery Protocols making them difficult to discover and consume. Therefore, to realize the future vision of *anytime, anywhere, any device*, there is an immediate need to address the issue of protocol heterogeneity to enable future services.

Recently, a lot of emphases have been given to address user satisfaction. This has led to end users demanding personalized services with better Quality of Experience. However, service personalization requires prior knowledge about users' preferences, device capabilities and network characteristics. Towards this, the use of context information has been very successful. Unfortunately, current Service Discovery Protocols have not been able to fully exploit its advantages.

Therefore, there is a need to combine the capabilities of service discovery with the advantages of context information to enable highly sophisticated services customized to end-user needs.

In this thesis, we proposed a new approach to discover services in the large. Our approach is based on protocol translation to enable service discovery irrespectively of their underlying Service Discovery Protocols. We introduce ZigZag, a middleware to reuse and extend current Service Discovery Protocols, designed for local networks, to discover available services across network boundaries as required in Future Internet. The ZigZag middleware can be deployed as a standalone solution or can be integrated into existing one thanks to its modular design. Furthermore, the middleware can be configured to discover services based on diverse application requirements with the help of *policies*. *Policies* enable developers to define system conditions so that ZigZag can aggregate service information as required by different applications. We tested our approach with the help of simulations and the results showed that ZigZag can both reduce the number of messages that flow through the network, and provide value sensitive information to the requesting entity.

Nevertheless, defining *policies* is a daunting task that would require developers to have an in depth understanding of ZigZag. To address this issue, we proposed a policy language to define policies in a clean and effective way. Moreover, the policy language has some added advantages in terms of dynamic management. Firstly, it can be used to delegate resources to other ZigZag nodes. Secondly, it can be used to dynamically configure policies on ZigZag nodes at runtime. Thirdly, it can be used to log system information. Additionally, the policy language can be used to define domain-specific actions, enabling developers to customize the policy language to address different domains.

6.1 Ongoing and Future work

Real scenario Testing
Since the proposed ZigZag solution has only been evaluated by simulations, the next step involves integrating the middleware in the Home-Box equipment in order to evaluate its performance in large-scale testbed platform envisioned in ALICANTE project. The latter is composed of four autonomous distributed heterogeneous network environments called *pilot islands*. These *pilot islands* are located at the premises of ALICANTE consortium members and support different media services. The first pilot island is located at the PT Inovação headquarters in Aveiro – Portugal; the second at the FTB headquarters in Beijing – China; the third at the UPB Campus in Bucharest – Romania; and the final at CNRS-LaBRI Campus in Bordeaux – France.

Privacy
The language support has provided a step forward to ZigZag by identifying different runtime conditions during the process of service discovery and personalization. However, there are limitations to the approach that we want to investigate further. We are currently investigating privacy,

a key concern in personalized service discovery. Furthermore, we plan to integrate and configure privacy-specific behavior using policies within the ZigZag middleware architecture.

Support of service invocation & delivery protocol
The current implementation of the ZigZag middleware only supports service discovery and service personalization in a heterogeneous environment. However, we are currently investigating how to extend our proposed architecture to support both service invocation and service delivery, so as to provide a full featured mechanism for service access in the context of Future Internet.

6.2 Concluding remarks

The ZigZag Middleware was originally designed to discovery media services in a multi-protocol heterogeneous environment. However, the advantages of ZigZag go beyond media access and can be effectively used to automate home and enable smart spaces. Furthermore, The recent improvement in technology and the popularity of *Internet of things* have created the right conditions for the reawakening of smart homes and smart spaces. Smart home include services like centralized control of lighting; HVAC (heating, ventilation and air conditioning) appliances; security locks of gates and doors and other systems. However, most vendors for these services relies on different protocols. Towards this, ZigZag can help vendors to coordinated the interaction between different services. Furthermore, ZigZag can also enable users' to control various services with their smartphones and tablets; to provide improved convenience and comfort from the palms of their hands. While the policy language was designed to help developers write policies for ZigZag middleware, its flexible design enables it to be used in other applications. For instance, in an e-mail application, the developers can load an e-mail plugin parser that would enable the policies to sort or filter e-mail messages in an inbox based on users' preferences.

Conclusion

Bibliography

[AAB+11] M. Alduan, F. Alvarez, J. Bouwen, G. Camarillo, P. Cesar, P. Daras, O. Festor, E. Izquierdo, N. Laoutaris, A.D. Mezaour, et al., *Future media internet architecture reference model (v1. 0)*, http://www.gatv.ssr.upm.es/nextmedia/images/fmiattreferencemodel.pdf, March 2011, NextMedia Project.

[AKA06] *AKARI Architecture Design Project*, http://www.nict.go.jp/en/photonic_nw /archi/akari/concept-design_e.html, May 2006, National Institute of Information and Communications Technology (NICT) of Japan.

[ALI13] *ALICANTE: mediA ecosystem depLoyment through ubIquitous Content-Aware NeTwork Environments*, http://www.ict-alicante.eu/, Feb 2010-2013, European FP7 Project.

[APS99] Mark Allman, Vern Paxson, and William Stevens, *RFC 2581: TCP congestion control*, http://tools.ietf.org/html/rfc2581, April 1999, IETF.

[AWSBL99] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, *The design and implementation of an intentional naming system*, ACM SIGOPS Operating Systems Review, vol. 33, ACM, 1999, pp. 186–201.

[BBK02] M. Balazinska, H. Balakrishnan, and D. Karger, *INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery*, Pervasive Computing (2002), 149–153.

[BCA+01] G.S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, et al., *The design and implementation of open orb 2*, IEEE Distributed Systems Online **2** (2001), no. 6, 1–40.

[BDD05] Ian S Burnett, Stephen J Davis, and Gerrard M Drury, *Mpeg-21 digital item declaration and identification-principles and compression*, Multimedia, IEEE Transactions on **7** (2005), no. 3, 400–407.

[BDS08] D. Benslimane, S. Dustdar, and A. Sheth, *Services mashups: The new generation of web applications*, Internet Computing, IEEE **12** (2008), no. 5, 13–15.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer, *Simple object access protocol (soap) 1.1*, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, May 2000, World Wide Web Consortium and others.

[BGR11] Yerom-David Bromberg, Paul Grace, and Laurent Réveillère, *Starlink: Runtime interoperability between heterogeneous middleware protocols*, Proceedings of the 2011 31st International Conference on Distributed Computing Systems (Washington, DC, USA), ICDCS '11, IEEE Computer Society, 2011, pp. 446–455.

[BI05] Y.-D. Bromberg and V. Issarny, *Indiss: Interoperable discovery system for networked services*, IFIP/ACM/Usenix International Middleware Conference, 2005, pp. 164–183.

[BKE03] Joseph Bauer, R Kutsche, and Rudiger Ehrmanntraut, *Identification and modeling of contexts for different information scenarios in air traffic*, Technische Universität Berlin, Diplomarbeit (2003).

[BLMM+94] T. Berners-Lee, L. Masinter, M. McCahill, et al., *RFC 1738: Uniform resource locators (URL)*, http://www.ietf.org/rfc/rfc1738.txt, Dec 1994, IETF.

[Blo70] B.H. Bloom, *Space/time trade-offs in hash coding with allowable errors*, Communications of the ACM **13** (1970), no. 7, 422–426.

[BM07] D. Brickley and L. Miller, *Foaf vocabulary specification 0.91*, http://xmlns.com/foaf/spec/20071002, Oct 2007, Tech. rep. ILRT Bristol.

[BMC+04] Paul Biron, Ashok Malhotra, World Wide Web Consortium, et al., *Xml schema part 2: Datatypes*, World Wide Web Consortium Recommendation REC-xmlschema-2-20041028 (2004).

[BNT10] Eugen Borcoci, Daniel Negru, and Christian Timmerer, *A novel architecture for multimedia distribution based on content-aware networking*, Communication Theory, Reliability, and Quality of Service (CTRQ), 2010 Third International Conference on, IEEE, 2010, pp. 162–168.

[BPSM+97] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau, *Extensible markup language (xml)*, World Wide Web Journal **2** (1997), no. 4, 27–66.

[BRLM09] Y.D. Bromberg, L. Réveillère, J. Lawall, and G. Muller, *Automatic generation of network protocol gateways*, Middleware 2009 (2009), 21–41.

[CFJ03] H. Chen, T. Finin, and A. Joshi, *An ontology for context-aware pervasive computing environments*, The Knowledge Engineering Review **18** (2003), no. 03, 197–207.

[Che11] Cheshire, S. and Krochmal, M., *Multicast DNS*, http://tools.ietf.org/html/draft-cheshire-dnsext-multicastdns-14, Feb 2011, IETF Internet-Draft.

[CHRT04] Luc Clement, Andrew Hately, Claus Riegen, and Rogers. Tony, *Universal Description Discovery and Integration Platform (UDDI) v3.0.2*, http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, Oct 2004.

[CK+00] Guanling Chen, David Kotz, et al., *A survey of context-aware mobile computing research*, Tech. report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[CK13] S. Cheshire and M. Krochmal, *RFC 6763: DNS-Based Service Discovery*, https://tools.ietf.org/html/rfc6763, Feb 2013, IETF.

[CNCS12] Soraya Ait Chellouche, Daniel Negru, Yiping Chen, and Mamadou Sidibe, *Homebox-assisted content delivery network for internet video-on-demand services*, 44–50.

[CPFJ04] H. Chen, F. Perich, T. Finin, and A. Joshi, *Soupa: Standard ontology for ubiquitous and pervasive applications*, Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual IEEE Conference on (2004), 258–267.

[CZH+99] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph, and R.H. Katz, *An architecture for a secure service discovery service*, Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, ACM, 1999, pp. 24–35.

[DAS01] Anind K Dey, Gregory D Abowd, and Daniel Salber, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*, Human–Computer Interaction **16** (2001), no. 2-4, 97–166.

[Dee88] S.E. Deering, *RFC 1054: Host extensions for IP multicasting*, http://tools.ietf.org/html/rfc1054, May 1988, IETF.

[DEFS98] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer, *Ontobroker: Ontology based access to distributed and semi-structured information*, Proceedings of the IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics-Semantic Issues in Multimedia Systems (Deventer, The Netherlands, The Netherlands), DS-8, Kluwer, B.V., 1998, pp. 351–369.

[Dro97] R. Droms, *RFC 2131: Dynamic Host Configuration Protocol (DHCP)*, http://www.ietf.org/rfc/rfc2131.txt, March 1997, IETF.

[ET11] ETSI-TS(MHM), *3gpp user data convergence (udc), framework for model handling and management*, http://www.3gpp.org/ftp/Specs/html-info/32181.htm, March 2011, ETSI TR 32 181 V10.0.0.

[ETS11a] ETSI-TS(CBIM), *3gpp user data convergence (udc), common baseline information model(cbim)*, http://www.3gpp.org/ftp/Specs/html-info/32182.htm, March 2011, ETSI TR 32 182 V10.0.0.

[ETS11b] ETSI-TS(St1), *3gpp generic user profile (gup), service requirements (stage 1)*, http://www.3gpp.org/ftp/Specs/html-info/22240.htm, May 2011, ETSI TS 122 240 V10.0.0.

[ETS11c] ETSI-TS(St2), *3gpp generic user profile (gup), requirements architecture (stage2)*, http://www.3gpp.org/ftp/Specs/html-info/23240.htm, May 2011, ETSI TS 123 240 V10.0.0.

[ETS11d] ETSI-TS(St3), *3gpp generic user profile (gup), network (stage 3)*, http://www.3gpp.org/ftp/Specs/html-info/29240.htm, May 2011, ETSI TS 122 240 V10.0.0.

[ETS11e] ETSI-TS(UDC), *3gpp user data convergence (udc)*, http://www.3gpp.org/ftp/Specs/html-info/22985.htm, March 2011, ETSI TR 22 985 V10.0.0.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616: Hypertext transfer protocol–HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, June 1999, IETF.

[FIP94] *European : Future Networks Projects*, http://cordis.europa.eu/home_en.html, 1994, Community Research and Development Information Service.

[G+93] Thomas R Gruber et al., *A translation approach to portable ontology specifications*, Knowledge acquisition **5** (1993), no. 2, 199–220.

[GBS03] P. Grace, G. Blair, and S. Samuel, *ReMMoC: A reflective middleware to support mobile client interoperability*, On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE (2003), 1170–1187.

[GL02] Michael Gruninger and Jintae Lee, *ONTOLOGY Applications and Design*, Communications of the ACM **45** (2002), no. 2, 39–41.

[GPVD99] E. Guttman, C. Perkins, J. Veizades, and M. Day, *RFC 2608: Service Location Protocol v2.0*, http://www.ietf.org/rfc/rfc2608.txt, June 1999, IETF.

[Gut02] E. Guttman, *RFC 3224: Vendor extensions for Service Location Protocol*, https://tools.ietf.org/html/rfc3224, Jan 2002, IETF.

[GVE00] A. Gulbrandsen, P. Vixie, and L. Esibov, *RFC 2782: A DNS RR for specifying the location of services (DNS SRV)*, https://tools.ietf.org/html/rfc2782, feb 2000, IETF.

[HIR03] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy, *Generating context management infrastructure from high-level context models*, In 4th International Conference on Mobile Data Management (MDM)-Industrial Track, Citeseer, 2003.

[HK02] S.Y. Ho and S.H. Kwok, *The attraction of personalized service for users in mobile commerce: an empirical study*, ACM SIGecom Exchanges **3** (2002), no. 4, 10–18.

[HMM08] Terry A Halpin, Antony J Morgan, and Tony Morgan, *Information modeling and relational databases*, Morgan Kaufmann, 2008.

[HSP+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger, *Context-awareness on mobile devices-the hydrogen approach*, System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, IEEE, 2003, pp. 10–pp.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi, *A policy based approach to security for the semantic web*, The Semantic Web-ISWC 2003, Springer, 2003, pp. 402–418.

[Kle08] J. Klensin, *RFC 5321: Simple mail transfer protocol*, https://tools.ietf.org/html/rfc5321, Oct 2008, IETF.

[KLW95] Michael Kifer, Georg Lausen, and James Wu, *Logical foundations of object-oriented and frame-based languages*, J. ACM **42** (1995), no. 4, 741–843.

[Kly99] Graham Klyne, *RFC 2533: A syntax for describing media feature sets*, http://xml2rfc.tools.ietf.org/html/rfc2533, March 1999, IETF.

[KNR02] Sanjeev Khanna, Joseph Seffi Naor, and Dan Raz, *Control message aggregation in group communication protocols*, Automata, Languages and Programming, Springer, 2002, pp. 135–146.

[KPS02] Charlie Kaufman, Radia Perlman, and Mike Speciner, *Network security: private communication in a public world*, Prentice Hall Press, 2002.

[LS+99] Ora Lassila, Ralph R Swick, et al., *Resource description framework (rdf) model and syntax specification*, http://www.w3.org/TR/1999/PR-rdf-syntax-19990105, Jan 1999, World Wide Web Consortium and others.

[LZL+11] Jianzhong LI, Yanping ZHANG, Zfanya Leib, Preston Rodrigues, Yiping Chen, Julien Arnaud, Daniel Négru, Eugen Borcoci, Pierre Bretillon, and Daniele Renzi, *D5.1.1 - SP/CP Service Environment - Service Management, Service Delivery*

- *I*, http://www.ict-alicante.eu/public/deliverables/alicante_d5.1.1_final.pdf, sept 2011, ALICANTE Consortium.

[MB87] Robert MacGregor and Raymond Bates, *The loom knowledge representation language.*, Tech. report, DTIC Document, 1987.

[MB98] John McCarthy and Sasa Buvac, *Formalizing context (expanded notes)*, http://www-formal.stanford.edu/jmc/mccarthy-buvac-98/context/context.html, 1998.

[MBH+04] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al., *Owl-s: Semantic markup for web services*, W3C Member submission **22** (2004), 2007–04.

[McC93] John McCarthy, *Notes on formalizing context*, http://www-formal.stanford.edu/jmc/context3/, 1993.

[MK09] Vipul Modi and Devon Kemp, *Web Service Dynamic Discovery (WSDD)*, http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.pdf, Aug 2009.

[MKP02] José M Martínez, Rob Koenen, and Fernando Pereira, *Mpeg-7: the generic multimedia content description standard, part 1*, Multimedia, IEEE **9** (2002), no. 2, 78–87.

[MM02] P. Maymounkov and D. Mazieres, *Kademlia: A peer-to-peer information system based on the xor metric*, Peer-to-Peer Systems (2002), 53–65.

[Moc87] P.V. Mockapetris, *RFC 1034: Domain names - concepts and facilities*, http://www.ietf.org/rfc/rfc1034.txt, nov 1987, IETF.

[Mog84] JC Mogul, *RFC 922: Broadcasting internet datagrams in the presence of subnets*, http://xml2rfc.tools.ietf.org/html/rfc922, Oct 1984, IETF.

[Mul12] Klaus Muller, *SimPy Simulator*, http://simpy.sourceforge.net/, April 2012, SimPy v2.3.

[MVH+04] D.L. McGuinness, F. Van Harmelen, et al., *Owl web ontology language overview*, W3C recommendation **10** (2004), 2004–03.

[NSF10] *NSF Future internet architecture project*, http://www.nets-fia.net/, Nov 2010, National Science Foundation USA.

[NSV+11] Dragos Niculescu, Mihai Stanciu, Marius Vochin, Eugen Borcoci, and Nikolaos Zotos, *Implementation of a media aware network element for content aware networks*, CTRQ 2011, The Fourth International Conference on Communication Theory, Reliability, and Quality of Service, 2011, pp. 78–83.

[Pas98] Jason Pascoe, *Adding generic contextual capabilities to wearable computers*, Wearable Computers, 1998. Digest of Papers. Second International Symposium on, IEEE, 1998, pp. 92–99.

[Per04] F Perich, *Mogatu bdi ontology*, University of Maryland, Baltimore County (2004).

[PFK+08] Alan Presser, Lee Farrell, Devon Kemp, William Lupton, et al., *Universal plug and play (upnp) device architecture v1.1*, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf, Oct 2008.

[PH04] F. Pan and J.R. Hobbs, *Time in owl-s*, Proceedings of AAAI-04 Spring Symposium on Semantic Web Services, 2004.

[PJYF03] Filip Perich, Anupam Joshi, Yelena Yesha, and Timothy Finin, *Neighborhood-consistent transaction management for pervasive computing environments*, Database and Expert Systems Applications, Springer, 2003, pp. 276–286.

[Pos80] J. Postel, *RFC 768: User datagram protocol*, http://xml2rfc.tools.ietf.org/html/rfc768, August 1980, IETF.

[RAH06] M. Raya, A. Aziz, and J.P. Hubaux, *Efficient secure aggregation in vanets*, Proceedings of the 3rd international workshop on Vehicular ad hoc networks, ACM, 2006, pp. 67–75.

[RBRN12] Preston Rodrigues, Yérom-David Bromberg, Laurent Réveillère, and Daniel Négru, *Zigzag: a middleware for service discovery in future internet*, Distributed Applications and Interoperable Systems, Springer, 2012, pp. 208–221.

[RCB+12] Preston Rodrigues, Soraya Ait Chellouche, Yérom-David Bromberg, Laurent Reveillere, and Daniel Négru, *Xtalk: A middleware for personalized service discovery in future internet*, Telecommunications and Multimedia (TEMU), 2012 International Conference on, IEEE, 2012, pp. 83–88.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM, 2001, pp. 161–172.

[RGCH09] Rajiv Ramdhany, Paul Grace, Geoff Coulson, and David Hutchison, *Manetkit: supporting the dynamic deployment and reconfiguration of ad-hoc routing protocols*, Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (New York, NY, USA), Middleware '09, Springer-Verlag New York, Inc., 2009, pp. 1:1–1:20.

[Riv92] R. Rivest, *RFC 1321: The MD5 message-digest algorithm*, http://www.ietf.org/rfc/rfc1321.txt, April 1992, IETF.

[RV06] R. Rajagopalan and P.K. Varshney, *Data-aggregation techniques in sensor networks: a survey*, Communications Surveys & Tutorials, IEEE **8** (2006), no. 4, 48–63.

[SAW94] Bill Schilit, Norman Adams, and Roy Want, *Context-aware computing applications*, Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on, IEEE, 1994, pp. 85–90.

[SB07] H. Saleet and O. Basir, *Location-based message aggregation in vehicular ad hoc networks*, Globecom Workshops, 2007 IEEE, IEEE, 2007, pp. 1–7.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen, *There is more to context than location*, Computers & Graphics **23** (1999), no. 6, 893–901.

[Sch95] William Noah Schilit, *A system architecture for context-aware mobile computing*, Ph.D. thesis, Columbia University, 1995.

[SCW+11] Mamadou Sidibe, Wael Cherif, Markus Waltl, Daniel Négru, and Roger Salgado, *The ALICANTE Home-Box Layer - I*, http://www.ict-alicante.eu/public/deliverables/alicante_d4.1.1_final.pdf, Sept 2011, ALICANTE Consortium.

[SGM02] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*, Addison-Wesley ISBN: 0-201-74572-0, 2002.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien, *A context modeling survey*, In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004.

[SLPF03] T. Strang, C. Linnhoff-Popien, and K. Frank, *Cool: A context ontology language to enable contextual interoperability*, Distributed applications and interoperable systems, Springer, 2003, pp. 236–247.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM, 2001, pp. 149–160.

[TAA+03] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.C. Hugly, E. Pouyoul, and B. Yeager, *Project jxta 2.0 super-peer virtual network*, Sun Microsystem White Paper. Available at www. jxta. org/project/www/docs (2003).

[Tap96] Don Tapscott, *The digital economy: Promise and peril in the age of networked intelligence*, vol. 1, McGraw-Hill New York, 1996.

[Tho04] Henry S Thompson, *Xml schema part 1: Structures second edition*, http://www.w3.org/TR/xmlschema-1/, Oct 2004, W3C Recommendation.

[WZGP04] X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung, *Ontology based context modeling and reasoning using owl*, Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, Ieee, 2004, pp. 18–22.

[YPG00] R. Yavatkar, D. Pendarakis, and R. Guerin, *RFC 2753: A Framework for Policy-based Admission Control*, http://www.ietf.org/rfc/rfc2753.txt, Jan 2000, IETF.

[ZNN+06] Bo Zhang, T. S. Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang, *Measurement based analysis, modeling, and synthesis of the internet delay space*, Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, IMC '06, ACM, 2006, pp. 85–98.

APPENDIX I: Publications

International Conferences

- Preston Rodrigues, Yérom-David Bromberg, Laurent Réveillère, and Daniel Négru. Zigzag: a middleware for service discovery in future internet. In *Proceedings of the 12th IFIP WG 6.1 international conference on Distributed Applications and Interoperable Systems*, DAIS'12, pages 208–221, Berlin, Heidelberg, 2012. Springer-Verlag.

- P. Rodrigues, S.A. Chellouche, Y.-D. Bromberg, L. Reveillere, and D. Negru. Xtalk: A middleware for personalized service discovery in future internet. In *Telecommunications and Multimedia (TEMU), 2012 International Conference on*, pages 83 –88, 30 2012-aug. 1 2012.

- Preston Rodrigues, Laurent Réveillère, Yérom-David Bromberg, and Daniel Négru. Scalable and interoperable service discovery for future internet. In *Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, M-MPAC '11, pages 3:1–3:7, New York, NY, USA, 2011. ACM.

Reports

- D5.1.1: SP/CP Service Environment - Intermediate. Jianzhong LI (RSN), Yanping ZHANG (FTB), Zfanya Leib (BANDWD), Preston Rodrigues, Yiping Chen, Julien Arnaud, Daniel Négru (CNRS-LaBRI), EugenBorcoci (UPB), Pierre Bretillon (TDF), Daniele Renzi (BSOFT).

- D5.2.1: The ALICANTE Service Registry - Intermediate. Zfanya Leib (BandWD), Jianzhong Li (RSN), Frederic Bouilhaguet (RSN), Preston Rodrigues (CNRS-LaBRI), Yiping CHEN (CNRS-LaBRI), Daniel Négru (CNRS-LaBRI), Jakub Gutkowski (PSNC), Lukasz Lopatowski (PSNC), Jordi Mongay Batalla (NIT), Mamadou Sidibé (VIOTECH), Evangelos Markakis (TEIC), Evangelos Pallis (TEIC)

APPENDIX I

APPENDIX II: Context-free Policy Language Grammar

A BNF-like description of the policy language The BNF-like syntax of the policy language is shown below. `typewriter` font is used for keywords. *italic* font is used for nonterminals. Parentheses are used for grouping. A superscript $*$ denotes 0 or more occurrences of the preceding item. A superscript $+$ denotes 1 or more occurrences of the preceding item. A superscript ? indicates that the preceding item is optional. Comments, begin with # and only after the actions.

policy_spec ::= `policy` *policy_condition* `then do action`
::= *policy_action* (`comment`)?
policy_condition ::= *conditions*
policy_action ::= *functions*
comment ::= `# comment or short policy desc`

conditions ::= *Simple* | *Complex*

Simple ::= *iden oper val*
iden ::= *letter* | (*letter*)*
oper ::= `<` | `!=` | `=` | `>` | `<=` | `>=`
val ::= *digits* | *letter*

Complex ::= *Simple op Simple* | `NOT` *Complex* |
::= *iden* `IN Collection {`*Complex*`}`
op ::= `AND` | `OR`

APPENDIX II

functions ::= *function* | (*functions*)$^{?}$
function ::= *Delegate* | *Configure* | *Logging* | *MEP*

Delegate ::= `delegate`(*delegateJob*, *delegateMessage*, *address*)
Configure ::= `config`(*policyOperation*, *policyInstance*, *address*, (*returnID*)$^{?}$)
Logging ::= `log`(*logMessage*)

MEP ::= *request-response* | *request-only*
request-response ::= `transmit_rr`(*parameters*)
request-only ::= `transmit_ro`(*parameters*)

parameters ::= *actionType*, *messageBody*, *address*, *messageID*

delegateJob ::= *aggregate* | *translation* | *adaptation*
actionType ::= *forward* | *context*

forward ::= `fwd`
aggregate ::= `aggr`
translate ::= `tran`

context ::= *context-user* | *context-device* | *context-network* | *context-session*
context_user ::= `con_u`
context_device ::= `con_d`
context_network ::= `con_n`
context_session ::= `con_s`

adaptation ::= *codec*
codec ::= *audio-codec* | *video-codec*
audio-codec ::= `acod`
video-codec ::= `vcod`

APPENDIX II

delegateMessage ::= `XML format`

messageBody ::= `XML format`

logMessage ::= *xalpha*

messageID ::= *alpha* | *digit*

returnID ::= *alpha* | *digit*

address ::= *httpaddress* | *fileaddress*

httpaddress ::= `http://`*hostport*(`/`*path*)$^{?}$(`?`*query*)$^{?}$
hostport ::= *host* (`:`*port*)$^{?}$
path ::= *void* | *segment*(`/`*path*)$^{?}$
query ::= *xalpha*

fileaddress ::= `file://`*host*`/`*path*
host ::= *hostname* | *hostnumber*
hostname ::= *alpha*(`.`*hostname*)$^{?}$
hostnumber ::= *ipv4* | *ipv6*
ipv4 ::= *digits*`.`*digits*`.`*digits*`.`*digits*
ipv6 ::= *hex*`::`*ipv4*
port ::= *digits*
alpha ::= *lalpha* | *halpha* | (*alpha*)$^{?}$
digits ::= *digit* | (*digits*)$^{?}$
xalpha ::= *alpha* | *digit*
xpalpha ::= *xalpha* | `+`
hex ::= *digit* | *af-alpha* | *AF-alpha*
lalpha ::= `a..z`
halpha ::= `A..Z`
digit ::= `0..9`
AF-alpha ::= `A..F`
af-alpha ::= `a..f`
void ::=
segment ::= *xpalpha*
reserved-keywords ::= `sdp-request-header | sdp-discovery-header |`
::= `search-key message-id | source-sdp |`
::= `target-sdp result | service-location`