Modèles, méthodes et outils pour les systèmes répartis multiéchelles

Sam Rottenberg

Thèse de doctorat de Télécom SudParis, préparée dans le cadre de l'école doctorale S&I, en accréditation conjointe avec l'Université d'Évry-Val-d'Essonne

Spécialité Informatique

Présentée par :

Sam ROTTENBERG

Modèles, méthodes et outils pour les systèmes répartis multiéchelles

Thèse présentée pour l'obtention du grade de
Docteur de Télécom SudParis

Soutenue le 27 avril 2015 devant le jury composé de :

Rapporteurs :		
Mme	Laurence Duchien	Professeur à l'Université de Lille 1
M.	Olivier Barais	Maître de Conférences HDR à l'Université de Rennes 1
Examinateurs :		
M.	Antoine Beugnard	Professeur à Télécom Bretagne
Mme	Chantal Taconet	Maître de Conférences HDR à Télécom SudParis - SAMOVAR (Directrice de thèse)
M.	Sébastien Leriche	Maître de Conférences à l'ENAC - Université de Toulouse (Encadrant)
Mme	Claire Lecocq	Directrice d'Études à Télécom SudParis - SAMOVAR (Encadrante)

Thèse numéro : 2015TELE0003

Résumé

Les systèmes informatiques sont des systèmes de plus en plus complexes, répartis sur plusieurs niveaux d'infrastructures des Technologies de l'Information et de la Communication (TIC). Dans ces systèmes interagissent des entités installées sur des petits équipements, comme des capteurs ou des systèmes RFID, mais aussi sur des systèmes puissants, installés sur des *clouds*, ainsi que sur des équipements intermédiaires comme les *smartphones* et les ordinateurs personnels. Ces systèmes sont parfois appelés des systèmes répartis multiéchelles. Le terme « multiéchelle » peut qualifier des systèmes répartis extrêmement variés suivant les points de vue dans lesquels ils sont caractérisés, comme la dispersion géographique des entités, la nature des équipements qui les hébergent, les réseaux sur lesquels elles sont déployées, ou encore l'organisation des utilisateurs. Pour une entité d'un système multiéchelle, les technologies de communication, les propriétés non fonctionnelles (en termes de persistance ou de sécurité), ou les architectures à favoriser, varient suivant la caractérisation multiéchelle pertinente définie ainsi que l'échelle à laquelle est associée l'entité. De plus, des architectures *ad hoc* de tels systèmes complexes sont coûteuses et peu durables.

Dans cette thèse, nous proposons un *framework* de caractérisation multiéchelle, appelé MuSCa. Ce *framework* inclut un processus de caractérisation fondé sur les concepts de points de vue, dimensions et échelles, permettant de mettre en avant, pour chaque système complexe étudié, ses caractéristiques multiéchelles. Ces concepts constituent le cœur d'un métamodèle dédié. Le *framework* que nous proposons permet aux concepteurs de systèmes répartis multiéchelles de partager une taxonomie pour qualifier chaque système. Le résultat d'une caractérisation est un modèle à partir duquel le *framework* produit des artefacts logiciels qui apportent, à l'exécution, la conscience des échelles aux entités du système. Nous validons qualitativement notre approche en présentant trois cas d'utilisation dans lesquels le *framework* MuSCa a été utilisé. Ces derniers proviennent de travaux de recherche réalisés pour la conception de *middleware* pour l'Internet des objets. Le premier cas d'utilisation est la caractérisation multiéchelle de scénarios de l'Internet des objets dans le but d'identifier les exigences liées au multiéchelle. Le deuxième cas d'utilisation est le déploiement d'entités logicielles dans un système réparti multiéchelle. Enfin, le troisième cas d'utilisation concerne le filtrage et le routage d'informations de contexte au sein d'un système réparti multiéchelle.

Mots-clés

Multiéchelle, Systèmes répartis complexes, Ingénierie dirigée par les modèles, *Middleware*.

Abstract

Computer systems are becoming more and more complex. Most of them are distributed over several levels of Information and Communication Technology (ICT) infrastructures. They may involve very small devices such as sensors and RFID, but also powerful systems such as cloud computers and knowledge bases, as well as intermediate devices such as smartphones and personal computers. These systems are sometimes referred to as multiscale systems. The word "multiscale" may qualify extremely various distributed systems according to the viewpoints in which they are characterized, such as the geographic dispersion of the entities, the nature of the hosting devices, the networks they are deployed on, or the users' organization. For one entity of a multiscale system, communication technologies, non-functional properties (in terms of persistence or security) or architectures to be favored may vary depending on the relevant multiscale characterization defined for the system and on the scale associated to the entity. Moreover, *ad hoc* architectures of such complex systems are costly and non-sustainable.

In this doctoral thesis, we propose a multiscale characterization framework, called MuSCa. The framework includes a characterization process based on the concepts of viewpoints, dimensions and scales, which enables to put to the fore the multiscale characteristics of each studied system. These concepts constitute the core of a dedicated metamodel. The proposed framework allows multiscale distributed systems designers to share a taxonomy for qualifying each system. The result of a characterization is a model from which the framework produces software artifacts that provide scale-awareness to the system's entities at runtime. We validate our approach qualitatively by presenting three use cases in which the MuSCa framework has been used. These use cases are taken from research works that aim at designing middleware for the Internet of things. The first use case is the multiscale characterization of scenarios for the Internet of things with the aim of identifying multiscale requirements. The second use case is the deployment of software entities in a multiscale distributed system. Finally, the third use case concerns the filtering and routing of context data in a multiscale distributed system.

Keywords

Multiscale, Complex distributed systems, Model-driven engineering, Middleware.

Remerciements

En premier lieu, je tiens à remercier Sébastien Leriche et Claire Lecocq de m'avoir encouragé à me lancer dans cette thèse, et Chantal Taconet d'avoir accepté de la diriger. Merci beaucoup à eux trois pour leur grande implication, leur soutien, leurs conseils, et surtout pour la confiance qu'ils m'ont accordée tout au long de ma thèse. J'ai eu grâce à eux le sentiment rassurant d'être encadré et guidé tout en gardant une grande sensation de liberté.

Je remercie également les membres du jury, en commençant par Laurence Duchien et Olivier Barais qui m'ont fait l'honneur d'accepter d'être rapporteurs de ma thèse. Je remercie aussi Antoine Beugnard de s'être intéressé à mes travaux et d'avoir accepté de faire partie du jury.

Je tiens ensuite à remercier toutes les personnes avec qui j'ai eu l'occasion de travailler au cours de cette thèse. Pour commencer, merci à l'ensemble des membres du département informatique de Télécom SudParis. Je ne peux malheureusement pas tous les citer ici. Merci à eux de m'avoir accompagné à travers des relectures d'articles, des conseils pour la soutenance, ou tout simplement des discussions autour d'un thé ! Je remercie spécialement Denis Conan pour tous ses conseils, Michel Simatic pour son écoute et ses paroles réconfortantes et encourageantes, et Olivier Berger qui m'a fait découvrir Org-mode [1], sans doute l'outil que j'ai le plus utilisé pendant cette thèse ! Merci aussi à Brigitte Houassine pour son aide dans toutes les démarches administratives et logistiques et pour sa capacité d'anticipation impressionnante. Je remercie également les membres du projet ANR INCOME d'avoir accordé un grand intérêt à mes travaux dès le départ et de m'avoir aidé à préciser mes propositions tout au long de mon travail.

Ces remerciements ne seraient pas complets sans parler de ceux qui m'ont accompagné dans ma « deuxième » vie : la danse. Je remercie tous mes professeurs de danse et en particulier Audrey et Fanny de m'avoir permis d'atteindre mes objectifs en parallèle de cette thèse. Merci aussi à mon coach Yannick de m'aider à garder la forme ! Leur présence m'a été précieuse bien au-delà de la danse et du sport.

Enfin je tiens à remercier mes proches pour leur soutien dans tout ce que j'ai entrepris pendant ces années de thèse. Merci à Hélène, Anthony(s), Ariane, Julie, Anaelle, Adrien, Sarah, Gertrude, et ceux que j'oublie. Merci aussi à Aurélien pour son soutien, sa présence, et parfois sa patience ! Et pour finir, un énorme merci à mes parents, et beaux-parents, de m'avoir toujours laissé libre de mes choix tout en me soutenant quoi qu'il arrive. (Et merci Maman pour la relecture complète du manuscrit !)

1. http://orgmode.org/

Table des matières

Table des figures

Liste des tableaux

Introduction

Chapitre 1

Introduction

Sommaire

1.1 Cadre général

Les progrès constants dans le domaine des Technologies de l'Information et de la Communication (TIC) permettent aux concepteurs de systèmes répartis de concevoir des systèmes innovants qui répondent à des besoins de plus en plus variés. En particulier, l'émergence du paradigme de l'Internet des objets ou des choses (*Internet of Things, IoT* en anglais) place l'utilisateur au cœur d'un environnement intelligent et connecté à l'Internet. En associant ce paradigme aux grandes capacités de stockage et de calculs offertes par le paradigme du *cloud computing*[1], ainsi qu'à des systèmes intermédiaires, comme les appareils mobiles ou les serveurs de proximité, les possibilités de conception de systèmes répartis distribués sur différents niveaux de l'infrastructure sont augmentées.

Cependant, la grande diversité des technologies existantes dans de tels systèmes complexes implique également une augmentation de la complexité du travail de concep-

1. Nous utilisons les expressions anglaises « cloud » et « cloud computing » qui sont plus couramment utilisées que les expressions françaises « nuage » et « informatique dans les nuages ».

tion, de développement et de déploiement de tels systèmes, en augmentant l'hétérogénéité des infrastructures sur lesquelles ils sont construits. De plus, ces systèmes peuvent être utilisés par des utilisateurs dispersés géographiquement et dont l'organisation sociale est également complexe.

Dans cette thèse, nous étudions ces systèmes répartis complexes dans le but de proposer des méthodes et des outils pour en réduire la complexité. En particulier, nous nous intéressons aux systèmes répartis complexes qui sont parfois qualifiés de systèmes répartis multiéchelles, car ils sont répartis à travers des équipements, des réseaux ou encore des zones géographiques de tailles très variées.

Nous pouvons noter que le concept de répartition multiéchelle est différent du concept de répartition large échelle. En effet, le terme large échelle a un sens quantitatif (grand nombre d'équipements, d'utilisateurs, de données à traiter, etc.), alors que le terme multiéchelle exprime une forte hétérogénéité [Satyanarayanan 1990], [Sandhu 1992]. L'hétérogénéité du système peut venir de différences de latences ou de protocoles des réseaux impliqués, de différences d'espace de stockage ou de nature des équipements, ou de variations de la dispersion des entités.

La notion de système complexe n'est pas réservée au domaine de l'informatique. Comme présenté dans [Chavalarias et al. 2009], un « système complexe » peut qualifier n'importe quel système composé d'un grand nombre d'entités hétérogènes, dans lequel des interactions locales entre entités créent différents niveaux de structures et d'organisations collectives. Il en est de même pour le concept d'échelle. Il est notamment utilisé en sciences physiques pour exprimer le fait que certaines lois s'appliquent uniquement à certaines échelles. Par exemple, à l'échelle nanoscopique (l'échelle atomique), les lois de la mécanique quantique s'appliquent, alors qu'à l'échelle macroscopique ce sont les lois de la mécanique classique qui s'appliquent.

Nous proposons, dans cette thèse, d'utiliser la notion d'échelle comme cadre d'étude pour obtenir une vision simplifiée d'un système réparti complexe. Nous proposons d'appliquer cette démarche aux systèmes répartis complexes qualifiés de multiéchelles en identifiant différentes échelles pertinentes dans ces systèmes. Nous pensons que, dans le domaine de l'informatique, les échelles présentes dans un système réparti complexe peuvent être vues comme des ensembles cohérents et homogènes qui peuvent être associés à des protocoles de communications, des architectures ou des comportements propres.

1.2 Problématique et démarche

Nous identifions la problématique suivante. La conception et le déploiement de systèmes répartis complexes sont des tâches difficiles qui demandent des outils pour les simplifier. En particulier, la forte hétérogénéité technique et la complexité de l'organisation des entités (sous-systèmes) peuvent amener les concepteurs et les développeurs à implémenter des solutions *ad hoc*, spécifiques à un système donné, qui ne peuvent pas être réutilisées pour la conception d'autres systèmes.

Nous nous sommes donc posé les questions suivantes : Comment faciliter la conception, le développement et le déploiement de systèmes répartis complexes ? En particulier, pouvons-nous proposer des méthodes et des outils qui permettent de concevoir des solutions adaptées non pas à un système donné mais à une famille de systèmes ? Notre proposition vise à exploiter et alimenter une base de connaissances et de technologies réutilisables pour de futurs systèmes.

Les principales caractéristiques des systèmes répartis complexes sont le grand nombre d'entités, la forte hétérogénéité technique, leur architecture dynamique, et les différents niveaux d'interactions entre entités. Nous remarquons que la complexité des systèmes répartis peut s'exprimer en termes d'échelles présentes dans ces systèmes, depuis différents points de vue (équipement, réseau, géographie, etc.). Les systèmes répartis complexes peuvent ainsi être vus comme des systèmes répartis multiéchelles. Cependant, des définitions, méthodes et outils, sont nécessaires pour mettre en œuvre cette nouvelle vision.

Par ailleurs, nous identifions qu'une approche de type descendante est appropriée pour maîtriser, voire réduire, la complexité de tels systèmes, et donc d'en faciliter la conception et le déploiement. Nous adoptons une des principales approches descendantes : l'Ingénierie Dirigée par les Modèles (IDM). Cette approche est intéressante car elle propose différents niveaux d'abstraction sous forme de modèles, et des transformations pour naviguer entre ces modèles. Cela permet aux concepteurs d'un système réparti complexe de maîtriser le niveau de complexité du système qu'ils conçoivent en sélectionnant un nombre restreint de concepts à prendre en compte dans la définition de chaque modèle. De plus l'IDM permet de rendre les modèles exploitables à l'exécution en proposant des mécanismes de génération de code à partir de modèles.

1.3 Contributions de la thèse

Dans cette thèse nous proposons de répondre à cette problématique en nous appuyant sur la définition d'une vision multiéchelle des systèmes répartis complexes. Cette vision, accompagnée d'un vocabulaire spécifique, nous permet d'étudier un système réparti complexe selon différents points de vue et en considérant les échelles qui le composent. Au lieu de considérer les entités du système une par une, nous les regroupons en échelles pertinentes pour ce système, ce qui permet de réduire l'hétérogénéité présente dans le système : de milliers d'entités hétérogènes, nous passons à quelques dizaines d'échelles différentes. Nous proposons ensuite de rendre cette vision multiéchelle exploitable en la modélisant et en y appliquant la démarche IDM. Cette démarche met à disposition des concepteurs de systèmes répartis des outils utilisables à la fois au cours de la conception du système (modélisation du système à l'aide des concepts du multiéchelle), et à l'exécution du système (génération d'artefacts exécutables à partir d'un modèle du système).

Ainsi, nous proposons le *framework*[2] MuSCa (***Mu**lti**S**cale **Ch**aracterization framework*) qui permet de caractériser la nature multiéchelle d'un système réparti complexe en identifiant les différentes échelles présentes dans le système.

La première contribution de cette thèse est un processus de caractérisation multiéchelle sur lequel s'appuie le *framework* MuSCa. Ce processus est principalement basé sur les concepts de points de vue, dimensions, et échelles. Une caractérisation multiéchelle peut à la fois s'appuyer et alimenter ce que nous appelons une taxonomie du multiéchelle qui regroupe tous les points de vue, dimensions et échelles déjà identifiés lors de caractérisations précédentes.

La deuxième contribution de cette thèse est la génération d'artefacts logiciels, appelés sondes multiéchelles, qui apportent la conscience des échelles aux entités d'un système en cours d'exécution. Grâce à ces artefacts, les entités du système deviennent conscientes de leur place dans l'organisation multiéchelle du système. Ce mécanisme de génération est implémenté dans le *framework* MuSCa.

Dans le futur, la caractérisation multiéchelle et les sondes multiéchelles pourront permettre aux concepteurs et développeurs de logiciels de construire, déployer et gérer

2. Nous utilisons le terme anglais « framework » qui est plus couramment utilisé que le terme français « cadriciel ».

des systèmes distribués complexes.

1.4 Scénario de validation

Cette section présente un scénario utilisé tout au long de ce manuscrit afin de motiver, d'illustrer et de valider l'approche de caractérisation multiéchelle des systèmes répartis complexes. Ce scénario a été élaboré dans le cadre du projet ANR INCOME [3], mené par des équipes de recherche localisées à Évry et à Toulouse. Ce scénario considère un système de gestion de contexte, qui est un système réparti complexe. Ce système est déployé à travers un grand nombre d'entités dans la ville de Toulouse. Le but de ce système est de permettre à un grand nombre d'utilisateurs finaux et d'objets connectés de partager leurs informations de contexte, comme par exemple leur géolocalisation. Le scénario est étudié selon deux aspects : l'aspect déploiement et l'aspect filtrage et routage de données de contexte.

Concernant le point de vue déploiement, le scénario propose d'exprimer trois exigences. Premièrement, il est nécessaire d'installer un composant logiciel dédié au filtrage de données de contexte dans chaque réseau local (LAN) contenant au moins un objet conscient du contexte. Ce composant doit être installé sur une machine possédant une bande passante supérieure à $50MB/s$. Deuxièmement, afin d'assurer la scalabilité de l'architecture du système, des composants dédiés au routage d'informations de contexte doivent être installés de façon hiérarchique, c'est-à-dire qu'il est nécessaire d'installer un composant dans chaque zone géographique de la ville de Toulouse : un par immeuble, un par quartier, et un pour la ville. Enfin, le composant de routage de la ville de Toulouse doit être installé dans un *cloud* et les composants conscients du contexte destinés aux utilisateurs finaux doivent être installés sur des *smartphones* [4].

Le deuxième aspect est l'aspect de filtrage et routage d'information de contexte. Le scénario propose de permettre à l'utilisateur d'exprimer des exigences concernant les informations de contexte qu'il ou elle souhaite recevoir et concernant les utilisateurs qui sont autorisés à recevoir des informations de contexte le ou la concernant. Le scénario considère les deux cas d'utilisation suivants. Dans le premier cas d'utilisation,

3. http://www.irit.fr/income/

4. Nous utilisons l'expression anglaise « smartphone » qui est plus couramment utilisée que l'expression française « téléphone intelligent ».

l'utilisatrice, que l'on appelle Sophie, est en train de se rendre au théâtre et souhaite trouver une place de parking pour sa voiture. Elle désire utiliser le système de gestion de contexte avec son *smartphone* afin de visualiser les places de parking accessibles à pied depuis le théâtre, où elle vient d'arriver. Dans le second cas d'utilisation, Sophie souhaite partager sa géolocalisation, mais seulement avec ses amis situés actuellement dans le même quartier qu'elle.

Ce scénario illustre le besoin d'exprimer différents types d'exigences en termes d'échelles : des exigences relatives à différentes échelles de distances (p. ex., accessible à pied, même quartier) entre des entités du système (p. ex., utilisateurs, places de parking), d'autres relatives à différentes échelles d'équipements (p. ex., *smartphone*, *cloud*), ou encore à différentes échelles de topologie réseau ou de zones géographiques administratives. Ces exigences en terme d'échelles dirigent les interactions entre les entités du système.

1.5 Plan du manuscrit

Après avoir introduit les travaux de cette thèse dans ce premier chapitre, la suite du document est composé de deux parties et d'une conclusion.

La première partie présente l'état de l'art. Elle est composée de deux chapitres. Le premier (chapitre 2) concerne les systèmes répartis complexes. Dans ce chapitre nous commençons par préciser la notion de systèmes répartis complexes, puis nous présentons des travaux de recherche récents qui impliquent des systèmes répartis complexes pour lesquels l'application d'une vision multiéchelle semble pertinente. Nous exposons ensuite les différents niveaux d'infrastructure qui composent ces systèmes répartis complexes. Enfin nous étudions différentes approches envisageables pour réduire la complexité de ces systèmes. Le deuxième chapitre de l'état de l'art (chapitre 3) présente les principes fondamentaux de l'approche IDM, que nous avons identifiée comme pertinente pour l'application d'une vision multiéchelle aux systèmes répartis complexes. Après avoir rappelé certaines notions générales sur les modèles, nous présentons l'approche IDM en elle-même, ainsi que des travaux de recherche qui proposent l'utilisation de cette approche dans le cadre des systèmes répartis complexes.

La deuxième partie du manuscrit constitue le cœur du document puisqu'elle pré-

sente les contributions de la thèse. Elle est constituée de trois chapitres. Les chapitres 4 et 5 présentent les deux contributions de la thèse et le chapitre 6 valide qualitativement ces contributions. Dans le chapitre 4, nous commençons par exposer les concepts du vocabulaire multiéchelle que nous proposons, ainsi que le processus de caractérisation multiéchelle. Puis nous précisons et formalisons ce processus à l'aide de concepts mathématiques de la théorie des ensembles. Dans le chapitre 5, nous présentons le *framework* MuSCa en détaillant les modèles sur lesquels est basé ce *framework* ainsi que les mécanismes de génération d'artefacts pour la conscience des échelles à l'exécution d'un système. Enfin, dans le chapitre 6, nous validons qualitativement notre démarche en présentant l'utilisation du *framework* MuSCa dans trois cas d'utilisations mis en œuvre dans le cadre du projet ANR INCOME.

Le dernier chapitre conclut le manuscrit et propose des perspectives aux travaux de la thèse.

Première partie

État de l'art – Systèmes Complexes et Modélisation

Chapitre 2

Systèmes répartis complexes

Sommaire

Dans ce chapitre, nous commençons, section 2.1, par discuter de la notion de systèmes répartis complexes. Puis, section 2.2, nous étudions différents travaux de recherches récents impliquant des systèmes répartis complexes, pour lesquels l'application d'une vision multiéchelle semble pertinente. Nous continuons, section 2.3, en présentant les différents niveaux d'infrastructure qui composent ces systèmes complexes. Puis, dans la section 2.4, nous présentons deux approches envisageables pour réduire la complexité de tels systèmes. Enfin, nous dressons un bilan de cette étude en section 2.5.

2.1 Systèmes complexes

Dans cette thèse, nous nous intéressons à des systèmes répartis qualifiés de complexes. Nous nous appuyons sur la définition générale suivante :

« *A "complex system" is in general any system comprised of a great number of heterogeneous entities, among which local interactions create multiple levels of collective structure and organization.* » [Chavalarias et al. 2009].

Cette définition s'applique aux systèmes complexes en général, y compris au-delà du domaine des Technologies de l'Information et de la Communication (TIC). Cette définition met en avant la forte hétérogénéité des entités présentes au sein des systèmes complexes. De plus, elle souligne la complexité de l'organisation de tels systèmes en termes d'interactions entre les nombreuses entités qui les composent.

Dans le domaine des TIC, un système réparti complexe est donc un système composé d'un grand nombre d'équipements fortement hétérogènes et dont l'architecture repose sur différents types d'interactions entre ces équipements : interactions locales ou distantes, différents protocoles de communication [Autili 2012, White 2012].

Cette définition générale peut être complétée par la nature ouverte, dynamique ou instable de l'architecture des systèmes répartis complexes [Autili 2012]. En effet, l'architecture d'un système réparti complexe ne peut pas être connue à l'avance, au cours de la phase de conception du système. Au contraire, des équipements peuvent apparaître et disparaître au cours de l'exécution, ce qui peut également entraîner des modifications de l'organisation du système, en termes d'interactions entre les équipements.

De façon complémentaire, un système réparti complexe peut être vu comme un système de systèmes [Tianfield 2008], c'est à dire comme un système composé de sous-systèmes autonomes qui interagissent entre eux et font émerger un comportement global du système.

Les auteurs de [White 2012] listent des problématiques spécifiques aux systèmes répartis complexes, principalement liées aux phases d'ingénierie système telles que la conception ou le déploiement. Par exemple, un des défis concerne le choix de l'équipement sur lequel un composant doit être déployé ainsi que sa configuration. Ces actions sont difficiles dans le cadre des systèmes répartis complexes à cause du grand nombre d'équipements présents et de leur forte hétérogénéité. Un autre défi concerne le respect d'exigences de qualité de service hétérogènes dans les différentes parties du système. Ces exigences doivent être respectées à la fois localement, dans chaque sous-système, et globalement. De plus, les systèmes répartis complexes nécessitent des modèles de programmation et de communication adaptés à leur hétérogénéité et leur dynamicité

intrinsèques : adaptabilité, composants faiblement couplés, etc. Enfin, l'application de procédures de certification de tels systèmes constitue un autre grand défi. En effet, il est très difficile d'évaluer et de tester l'exécution d'un système réparti complexe en condition réelle.

Après avoir identifié les principales caractéristiques liées aux systèmes répartis complexes, nous présentons, dans la section suivante, des cas d'utilisation représentatifs de tels systèmes.

2.2 Cas d'utilisation de systèmes répartis complexes hétérogènes

Plusieurs travaux de recherche récents présentent des systèmes répartis complexes [Kessis 2009], [Satyanarayanan 2011], [van Steen 2012], [Blair 2012], [Arcangeli 2012]. Certains de ces systèmes sont explicitement décrits par leurs auteurs comme multiéchelles [Kessis 2009], [Blair 2012], [Arcangeli 2012]. Dans la section 2.2.1, nous décrivons certains de ces travaux afin de mettre en lumière une vision multiéchelle des systèmes présentés. Puis, dans la section 2.2.2, nous dressons un bilan du sens de la vision multiéchelle pour les systèmes répartis complexes.

2.2.1 Quelques systèmes et cas d'utilisation multiéchelles

Dans cette section, nous présentons quelques cas d'utilisation multiéchelles mettant en évidence la diversité des aspects équipement, réseau, utilisateur et géographie.

2.2.1.1 Surveillance de crue

Blair et Grace [Blair 2012] présentent un scénario de surveillance de l'environnement et en particulier de surveillance de crue d'une rivière. En décrivant le scénario, les auteurs utilisent le terme d'« échelle » pour désigner des équipements de tailles différentes. En effet, des capteurs sont utilisés pour surveiller une rivière, les informations obtenues de ces capteurs sont envoyées vers le Cloud car le traitement de ces informations nécessite une très grande puissance de calcul. Les informations scientifiques

ainsi que les alertes d'urgence sont accessibles depuis les équipements mobiles de différents acteurs (scientifiques, services d'urgence, grand public). La figure 2.1 résume les différents types d'équipements impliqués dans ce scénario.

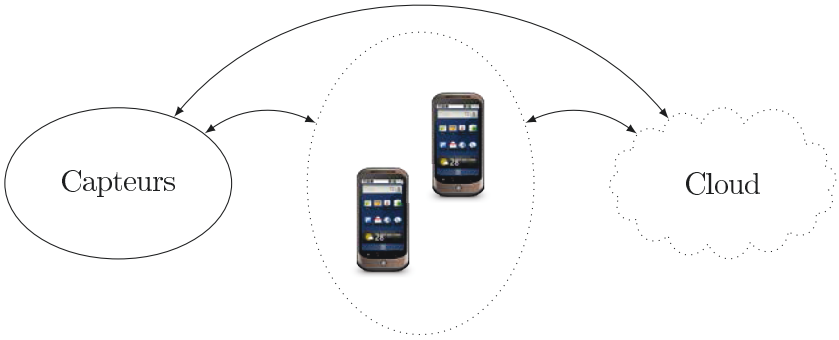

FIGURE 2.1 – Équipements impliqués dans la surveillance de crue

Ce scénario est un exemple proposé par les auteurs pour illustrer les nouveaux défis liés aux systèmes multiéchelle. La principale difficulté technique identifiée par les auteurs est l'extrême hétérogénéité technique présente dans un tel système. Cette hétérogénéité en terme d'APIs (Application Programming Interfaces), de systèmes d'exploitations (pour les *smartphones*), ou de paradigmes de programmation est telle que l'idée d'une solution standard est écartée par les auteurs. D'après eux, toute solution standard sera rapidement obsolète ou incompatible avec la complexité liée à une situation multiéchelle. Ils proposent donc le développement d'intergiciels « émergents » qui seraient générés à l'exécution pour former un pont entre les différentes parties de l'application.

2.2.1.2 Gestion de ressources en contexte multiéchelle

Kessis et al. [Kessis 2009] proposent un intergiciel de gestion de ressources dans un contexte multiéchelle. Le but de ce système est de proposer une gestion flexible d'un ensemble de ressources réparties de façon multiéchelle. Dans cet article, le terme multiéchelle est employé pour décrire la répartition des ressources à travers des réseaux de tailles différentes : PAN (Personal Area Network), LAN (Local Area Network), WAN (Wide Area Network).

La solution proposée consiste à regrouper les ressources en domaines et fédérations

de domaines afin d'implémenter différents modèles de gestion. Il est également possible de passer d'un modèle de gestion à un autre au cours de l'exécution.

Une représentation architecturale est extraite des ressources gérées afin de créer une couche d'abstraction homogène qui représente des ressources hétérogènes. Cette couche est composée d'éléments basiques qui peuvent être assemblés pour former des éléments composites. Les éléments basiques et composites sont tous considérés comme des éléments gérés de façon homogène du point de vue de l'application.

À partir de ces deux principes de gestion (domaines et éléments), l'intergiciel DASIMA a été développé. Il implémente une gestion flexible de ressources car il permet aux utilisateurs de changer le modèle de gestion mis en place et la granularité de l'application de gestion au cours de l'exécution.

2.2.1.3 Internet des objets et gestion de contexte

Le projet ANR INCOME [Arcangeli 2012] a pour objectif de proposer une infrastructure de gestion de contexte multiéchelle pour l'Internet des objets. Le concept même d'Internet des objets possède certains aspects multiéchelles. Ce concept fait intervenir des équipements de très petites tailles (capteurs, effecteurs, etc.) qui communiquent entre eux tout en étant répartis sur l'ensemble de l'Internet. L'entité logicielle responsable de la collecte, de la gestion (traitement et filtrage) et de la présentation des informations de contexte aux applications est le gestionnaire de contexte. C'est une entité essentielle pour le développement des applications sensibles au contexte grand public. La gestion de contexte est un problème traité généralement dans le cadre de réseaux ambiants. Dans le cadre de réseaux multiéchelles (ambiant, Internet, nuages) et au-dessus de l'Internet des objets, la gestion de contexte devient autrement plus complexe. Elle doit prendre en compte l'hétérogénéité des données, répartir les traitements et les flux d'informations, assurer le passage à l'échelle, gérer des informations de qualité de contexte pour permettre des prises de décision appropriées, respecter la vie privée lors de la transmission des informations de contexte, s'adapter à des environnements dynamiques, ou encore identifier des situations en mixant des données issues de l'Internet des objets et de bases de connaissances.

Le projet INCOME a pour ambition de réaliser de la gestion de contexte multiéchelle dans lequel des traitements seront à la fois réalisés sur des petits équipements au

plus près de leur production, mais également sur des *clouds* pour bénéficier de véritables bases de connaissances et de puissance de calcul. Les consommateurs des informations de contexte produites seront quant à eux répartis sur l'ensemble de l'Internet. De plus, le système devra être capable de prendre en compte la dispersion géographique des entités du système (utilisateurs ou objets), c'est-à-dire de permettre différents types d'interactions entre entités en fonction de leur localisation : interactions entre entités en environnement ambiant, interactions locales ou peu distantes, ou encore interactions très distantes. Enfin, le système prendra en compte l'organisation sociale de ses utilisateurs, c'est-à-dire les groupes et communautés d'utilisateurs.

2.2.1.4 Serveurs de proximité

Le concept de *cloudlet* est évoqué dans [Satyanarayanan 2011] et [Satyanarayanan 2009]. Les auteurs expliquent que, d'une part, les équipements mobiles seront toujours limités en terme de puissance de calcul et devront donc toujours déléguer une partie de leurs calculs, et que, d'autre part, le *cloud computing* n'est pas forcément la meilleure solution car il pose des problèmes de latence non négligeables.

La solution proposée est la mise en place de serveurs de proximité sans état et connectés à Internet (*cloudlets*) auxquels les équipements mobiles peuvent se connecter directement via un réseau Wi-Fi. Ces équipements sont déployés comme des *hotspots* Wi-Fi dans des cafés, magasins etc. Ils possèdent une forte puissance de calcul et permettent aux équipements mobiles d'externaliser les calculs trop complexes sur une machine proche, ce qui résout les problèmes de latence du *cloud computing*.

Un scénario proposé dans cet article est celui de l'aide aux personnes atteintes de la maladie d'Alzheimer grâce à des lunettes à réalité augmentée. Ces lunettes possèdent une caméra pour filmer ce que voit l'utilisateur et des écouteurs pour lui retourner des informations. À partir d'algorithmes comme la reconnaissance faciale ou le traitement d'informations de contexte, il serait possible de dire à l'utilisateur le nom de la personne qui est en face de lui, de lui rappeler de sortir son chien ou encore d'arroser ses plantes. Ce scénario fait intervenir des capteurs pour obtenir des informations de contexte, une base de connaissance distante et des *cloudlets* afin d'améliorer la performance et de pouvoir rapidement traiter les informations de contexte. Nous sommes donc bien en présence d'un scénario multiéchelle car il fait intervenir des équipements de très petites

tailles (capteurs) dépourvus de puissance de calcul, des *cloudlets* accessibles via un réseau local, et une base de connaissance sur Internet.

2.2.2 Bilan sur la vision multiéchelle des systèmes répartis complexes

L'étude précédente nous montre que les termes « échelle » et « multiéchelle » sont utilisés dans certains travaux de recherche récents afin de décrire l'architecture répartie de systèmes distribués complexes. Dans d'autres travaux, comme dans le cas des *cloudlets*, ces termes ne sont pas utilisés explicitement mais la vision multiéchelle s'applique également. Cependant, aucun de ces travaux ne propose une définition uniforme de la vision *multiéchelle* des systèmes distribués ni d'information concernant l'ensemble des cas pertinents pour son application.

Nous pouvons cependant remarquer que la nature multiéchelle d'un système réparti complexe peut être relative à différents points de vue. En effet, certains auteurs introduisent le terme d'échelle pour mettre en avant la forte hétérogénéité technique de l'architecture d'un système en terme d'équipements, d'autres en terme de réseaux, ou encore de données. De plus, un système peut être vu comme multiéchelle depuis des points de vue moins techniques qui concernent, par exemple, la dispersion géographique des entités ou des utilisateurs du système, ou l'organisation sociale des utilisateurs.

Un des objectifs de cette thèse est de fournir une formalisation homogène pour caractériser les systèmes répartis multiéchelles.

Dans la section suivante, nous nous intéressons à l'étude d'un système réparti multiéchelle en étudiant différents niveaux d'infrastructure.

2.3 Infrastructures pour les systèmes répartis complexes : de la *smartdust* au *cloud*

La figure 2.2 présente notre vision des différents niveaux d'infrastructure des TIC qui composent les systèmes répartis complexes. Cette figure considère quatre niveaux de TIC : en bas se trouve le niveau de la *smartdust*[1], de la *Radio Frequency Identification*

1. Nous utilisons le terme « smartdust » qui est plus couramment utilisé que le terme français « poussière intelligente ».

(RFID), et des capteurs intelligents ; en haut se trouve le niveau du *cloud computing* ; et entre les deux se trouvent deux niveaux d'infrastructure intermédiaires : premièrement les équipements personnels et deuxièmement les *cloudlets* [Satyanarayanan 2009] déployés dans des cafés ou arrêts de bus, par exemple.

Un système comme celui présenté figure 2.2 possède la topologie d'un système complexe. Les interactions entre les entités du système sont décentralisées. Beaucoup d'interactions sont locales, mais certaines ont lieu entre différents niveaux d'infrastructure. C'est pourquoi le graphe d'interaction entre les entités est non trivial et difficile à simplifier. De plus, ces systèmes complexes sont composés d'un grand nombre d'entités hétérogènes, qui collaborent à travers différents réseaux, protocoles, règles, en fonction de leur organisation ou de leur dispersion géographique, par exemple.

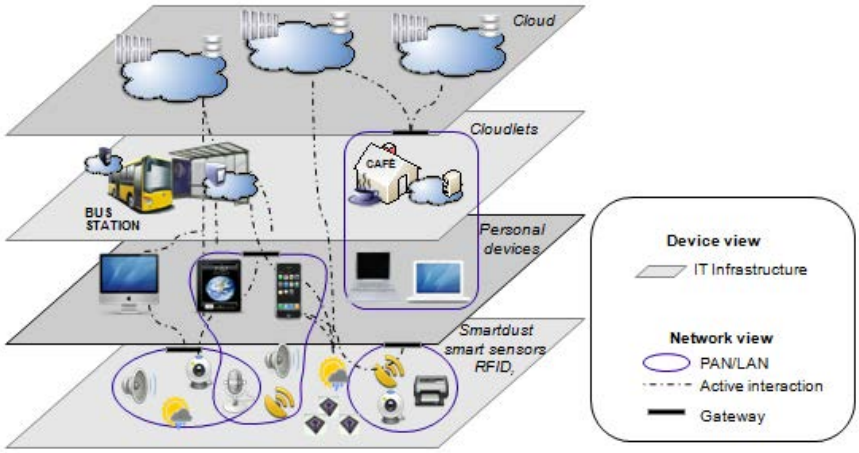

FIGURE 2.2 – Niveaux d'infrastructure des TIC d'un système réparti multiéchelle

Dans les trois sections suivantes, nous détaillons les différents niveaux de TIC que nous avons identifiés précédemment : premièrement le niveau de la *smartdust*, des systèmes RFID, des capteurs intelligents et de l'Internet des objets, puis le niveau du *cloud computing*, et enfin les niveaux intermédiaires.

2.3.1 *Smartdust*, RFID, capteurs intelligents et Internet des objets

La *smartdust* est une proposition de la DARPA de 1997 qui est parvenue à construire des nœuds de capteurs sans fil d'un millimètre cube de volume [Warneke 2001]. Le

concept *smartdust*, qui est employé pour désigner des ordinateurs minuscules, est l'un des piliers de l'Internet des objets. Il a été complété par les systèmes RFID, les réseaux de capteurs sans fil (*Wireless Sensor Networks*, WSNs), et les réseaux d'objets intelligents (*Smart Objects Networks*, SONs), pour constituer les technologies principales d'infrastructure et de communication qui structurent l'Internet des objets [Miorandi 2012].

Le concept d'Internet des objets a pour but d'étendre l'Internet classique aux objets physiques du monde réel. Ainsi, une grande quantité d'objets informatiques représentant des objets du monde réel peuvent être connectés, temporairement ou de manière permanente, à l'infrastructure réseau mondiale.

À l'origine, l'Internet des objets a été pensé comme la capacité d'intégrer automatiquement aux systèmes d'information des données relatives à l'identité, la localisation et l'état de certaines entités physiques ou conditions environnementales. Plus récemment, le paradigme a évolué vers une architecture qui étend celle des systèmes RFID en réseau pour lui apporter de nouvelles fonctionnalités de coopération entre objets intelligents. Ce nouveau paradigme est appelé le *Machine To Machine*, M2M. Il repose sur des technologies comme la communication en champ proche (*Near Field Communication*, NFC), les réseaux de capteurs sans fil, la communication par courants porteurs de ligne (CPL), ou les réseaux mobiles. Pour illustrer cette évolution, Chen affirme dans [Chen 2012] :

« *In the future, digital sensing, communication, and processing capabilities will be ubiquitously embedded into everyday objects, turning them into the Internet of Things (IoT, or machine-to-machine, M2M). In this new paradigm, smart devices will collect data, relay the information or context to each another, and process the information collaboratively using cloud computing and similar technologies. Finally, either humans will be prompted to take action, or the machines themselves will act automatically.* »

Cette dernière citation propose une évolution de la définition de l'Internet des objets en y intégrant le M2M. Elle affirme que les objets —ou choses— ne sont plus uniquement considérés comme des producteurs d'information. Ce sont des acteurs indépendants qui peuvent également échanger, stocker et traiter des informations de manière autonome, ainsi qu'interagir entre eux pour réaliser de manière coopérative des actions pour les utilisateurs humains, sans nécessairement les impliquer dans ce processus. De plus, cette définition introduit un aspect multiéchelle (des capteurs au *cloud*), qui est l'une des prochaines grandes étapes dans l'évolution de l'Internet des objets. Comme le

mentionnent les auteurs de [Gluhak 2011], l'évolution depuis des réseaux de capteurs sans fil isolés (l'Internet des objets d'aujourd'hui) à une infrastructure réseau mondiale (l'Internet des objets de demain) ouvre de nouveaux défis.

2.3.2 *Cloud computing*

Le *National Institute of Standards and Technology* (NIST) propose la définition suivante du *cloud computing* [Mell 2011] :

« *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.* »

Le NIST identifie cinq caractéristiques essentielles qui composent le modèle de *cloud* : libre-service à la demande, accès au réseau global, mise en commun des ressources, élasticité, et service mesuré.

Dans le contexte de l'Internet des objets, la coopération entre les objets intelligents et les *clouds* pourrait être profitable. L'accès au réseau global, l'élasticité et la mise en commun des ressources sont les principales caractéristiques qui font du *cloud* un environnement informatique intéressant pour coopérer avec des équipements plus limités. En effet, l'Internet des objets étant composé d'une très grande quantité d'objets, c'est un paradigme qui implique la génération d'un énorme volume de données, et qui nécessite une puissance de calcul importante pour l'analyse de ces données. L'élasticité du *cloud computing* est donc une caractéristique intéressante. De plus, dans le but de fournir un accès ubiquitaire aux données, l'accès au réseau global est une caractéristique indispensable.

2.3.3 Entre les objets et les *clouds*

Jusqu'à présent, nous avons décrit deux niveaux extrêmes d'infrastructure de TIC des systèmes répartis complexes : les objets communicants, et les *clouds*. Nous détaillons maintenant des équipements intermédiaires, comme les passerelles de l'Internet des

objets, les équipements mobiles et les serveurs fixes, qui sont également présents dans les systèmes répartis complexes.

Certains réseaux de l'Internet des objets, comme les réseaux de capteurs, utilisent des protocoles réseaux spécifiques (p. ex., ZigBee [2]). Par conséquent, des ordinateurs dédiés, appelés passerelles, sont nécessaires afin d'interconnecter ces réseaux aux réseaux d'échelle supérieure.

Les équipements mobiles, comme les téléphones mobiles, les tablettes, et les ordinateurs portables sont couramment utilisés par les utilisateurs nomades. Ils sont essentiels à la construction de services centrés sur l'utilisateur. Comme les ressources de ces équipements mobiles restent faibles comparées à celles des serveurs fixes, la coopération entre les équipements mobiles et les *clouds* est une approche prometteuse. Ainsi, dans [Satyanarayanan 2001], Satyanarayanan introduit le concept de *cyber foraging* [3] dans lequel les équipements mobiles déchargent une partie de leurs traitements lourds à des ordinateurs plus puissants.

Afin de réduire les problèmes de latence, Satyanarayanan propose, dans [Satyanarayanan 2009], une coopération entre des équipements mobiles et des *clouds* de proximité, appelés *cloudlets*. D'une part, les équipements mobiles doivent pouvoir déléguer une partie de leurs traitements à des serveurs. Et d'autre part, les serveurs distants accessibles dans le *cloud* ne constituent pas une solution optimale car les interactions dans des réseaux WAN entraînent une latence non négligeable. La solution proposée consiste en un déploiement de *clouds* de proximité. Ces *cloudlets* sont sans état, connectés à Internet, et sont destinés à être déployés sur le modèle des points d'accès Wi-Fi dans des cafés, magasins, arrêts de bus, etc. Ils permettraient de fournir une puissance de calcul élevée, permettant aux équipements mobiles de décharger une partie de leurs traitements avec une faible latence.

Après avoir décrit quelques cas d'utilisation mettant en avant l'aspect multiéchelle des systèmes répartis complexes, et avoir décrit les niveaux d'infrastructure rentrant en jeu dans ces systèmes, nous nous interrogeons, dans la section 2.4, sur les méthodes permettant d'analyser les systèmes répartis complexes.

2. Norme IEEE 802.15.4

3. Nous utilisons l'expression anglaise « cyber foraging » qui est plus couramment utilisée que l'expression française « butinage informatique ».

2.4 Analyse des systèmes complexes

Comme nous l'avons vu dans les sections précédentes, les systèmes répartis complexes sont des systèmes composés d'entités très nombreuses et fortement hétérogènes, et dont l'architecture est dynamique. Ces caractéristiques rendent la conception et le déploiement de tels systèmes très complexe. C'est pourquoi, dans cette section, nous présentons deux approches envisageables pour simplifier cette complexité. La première est l'approche ascendante, qui consiste à construire une vision simplifiée d'un système à l'exécution, et la seconde est l'approche descendante, qui consiste à construire une vision simplifiée dès la conception du système.

L'approche ascendante étudie un système réparti complexe réel à l'exécution. Certains travaux récents proposent des algorithmes pour étudier des patrons de collaboration entre des systèmes complexes réels. Par exemple, les auteurs de [Pons 2011] proposent un algorithme de détections de communautés afin de trouver des structures de communautés pour partitionner un système complexe. Ces algorithmes étudient la structure hiérarchique des systèmes complexes réels. Cependant, à cause de la nature à la fois large échelle, dispersée géographiquement et dynamique des systèmes complexes, une analyse multiéchelle à l'exécution est difficile à réaliser.

La deuxième approche est l'approche descendante, qui étudie un système réparti complexe dès la phase de conception. Il est nécessaire de proposer de nouvelles approches pour permettre aux développeurs de systèmes répartis complexes de travailler à un haut niveau d'abstraction. Dans l'article [France 2007], les auteurs proposent d'utiliser l'Ingénierie Dirigée par les Modèles (IDM) pour faciliter le développement des systèmes complexes. Ils argumentent que développer des systèmes répartis complexes sans utilisation de l'IDM augmente l'utilisation de technologies centrées sur le code, et empêche les développeurs de se concentrer sur les besoins fonctionnels, non fonctionnels et architecturaux du système. Cela produit des systèmes *fabriqués à la main*, fortement couplés aux technologies, et donc non maintenables ni évolutifs.

2.5 Conclusion

Dans ce chapitre, nous avons commencé par présenter les systèmes répartis complexes de façon générale afin d'en déterminer les principales caractéristiques. Nous

retenons, dans le cadre de cette thèse, le terme de système réparti complexe pour représenter un système ayant les caractéristiques suivantes :

— grand nombre d'entités,
— hétérogénéité,
— dynamicité,
— organisation complexe (différents types d'interactions entre entités).

Nous avons présenté différents cas d'utilisation qui font intervenir des systèmes répartis complexes en remarquant que plusieurs auteurs qui travaillent sur ces systèmes utilisent les termes d'« échelle » et de systèmes « multiéchelles » afin de caractériser la complexité de ces systèmes. Ces cas d'utilisation ont permis d'identifier qu'un système complexe peut être multiéchelle en considérant des points de vue techniques (p. ex., équipement, réseau) ou non technique (p. ex., géographie).

Nous avons présenté ensuite en détails les différents niveaux d'infrastructure des TIC sur lesquels sont construits les systèmes répartis complexes. Cette partie a permis d'identifier les principaux éléments techniques (équipements et réseaux) constitutifs de ces infrastructures et leurs coopérations.

Enfin nous avons distingué deux approches qui peuvent permettre de réduire la complexité de la gestion de tels systèmes : les approches ascendantes et descendantes. Nous concluons ce chapitre en considérant que l'approche descendante est la plus adaptée afin de disposer d'une vision simplifiée d'un système complexe dès sa phase de conception. C'est pourquoi le chapitre suivant est consacré à l'une des principales approches descendantes pour la conception de systèmes informatiques : l'ingénierie dirigée par les modèles.

Chapitre 3

Ingénierie dirigée par les modèles

Sommaire

Dans la section 2.4 (chapitre précédent), nous avons identifié qu'une approche descendante était la plus pertinente concernant l'application d'une vision multiéchelle aux systèmes répartis complexes. Au sein des approches descendantes, l'Ingénierie Dirigée par les Modèles (IDM) est la plus complète et la plus utilisée. C'est pourquoi, dans cette section, nous introduisons les principes fondamentaux de l'IDM, sur lesquels nous nous sommes basés pour la contribution de cette thèse.

Nous commençons, dans la section 3.1, par rappeler ce qu'est un modèle, et ce qu'apporte la modélisation dans différents domaines scientifiques ou techniques. Dans cette première section nous présentons l'activité de modélisation telle qu'elle est utilisée la plupart du temps : comme un moyen de réflexion et de communication entre êtres humains. Cette étape est nécessaire afin d'introduire, dans la section 3.2, l'évolution proposée par l'approche de l'IDM dans le domaine de l'informatique. En effet, cette approche propose des méthodes de conception d'applications et de systèmes informatiques

centrées sur les modèles, qui deviennent alors des outils productifs destinés à être compris non seulement par les êtres humains mais également par les outils informatiques. Enfin, en section 3.3, nous envisageons l'utilisation de l'IDM pour la conception de systèmes répartis complexes.

3.1 Introduction à la modélisation

Nous commençons ce chapitre par la définition de l'activité de modélisation (section 3.1.1), l'introduction des langages de modélisation (section 3.1.2), et l'utilisation de ces modèles (section 3.1.3).

3.1.1 Une représentation simplifiée du monde réel

L'être humain cherche, depuis toujours, à étudier et à comprendre le monde qui l'entoure. Cependant ce monde est souvent bien trop complexe et imprévisible pour pouvoir être étudié tel quel. Il y a trop d'exceptions, de phénomènes contradictoires et de paramètres à prendre en compte. C'est là qu'intervient la modélisation.

L'activité de modélisation consiste à décrire une entité réelle, en ne retenant que certaines caractéristiques nécessaires pour répondre à un problème donné [Jézéquel 2012]. Le résultat de cette activité est un modèle de l'entité modélisée. Par définition, tout modèle est une simplification et donc une représentation en partie erronée du monde réel. Cependant, un modèle est également une abstraction plus générique que l'entité modélisée, ce qui permet d'effectuer des raisonnements théoriques sur un modèle et de les appliquer à plusieurs entités réelles, sans avoir à prendre en compte toutes les spécificités de chaque entité. Il est important de noter qu'un modèle est, au départ, destiné à être dressé, compris et utilisé par l'être humain. Un modèle doit donc être suffisamment précis pour représenter fidèlement le monde réel, mais également suffisamment simple pour pouvoir être utilisé par l'être humain dans le but de discuter d'un problème et d'y répondre, ou d'élaborer et de vérifier des théories de façon générique.

Les domaines d'application de la modélisation sont très variés : il est possible de modéliser des phénomènes physiques ou chimiques, ou encore des tendances économiques sous forme de lois mathématiques, des constructions sous forme de plans en deux ou

trois dimensions, ou encore des processus de fabrication ou de gestion sous forme de diagrammes d'activités.

3.1.2 Les langages de modélisation

Un modèle peut prendre différentes formes : une équation mathématique, une représentation textuelle, ou encore une représentation graphique (courbe, diagramme, plan, etc.). Cependant toutes ces représentations ont en commun le fait d'être définies par un langage spécifique. Un langage de modélisation définit l'ensemble des concepts utilisables pour dresser un modèle d'une entité réelle. Comme nous l'avons vu plus haut, un modèle doit être à la fois précis et suffisamment générique. C'est en fait le langage de modélisation qui garantit ces propriétés. Un langage de modélisation doit donc être composé de suffisamment de concepts pour pouvoir décrire des modèles précis et fidèles au monde réel, mais sans devenir trop complexe pour que les modèles restent simples et exploitables. Par conséquent, il est préférable de définir des langages de modélisation spécifiques pour chaque aspect du monde réel que l'on cherche à modéliser, quitte à devoir dresser plusieurs modèles pour une même entité réelle.

Au delà des concepts proposés, un deuxième aspect fondamental d'un langage de modélisation est sa représentation. En effet, un langage doit pouvoir non seulement décrire efficacement une entité réelle mais également proposer une représentation claire qui permette une compréhension rapide du modèle par l'être humain. Que cette représentation soit textuelle ou graphique, le langage doit être facilement lisible et bien documenté, afin de pouvoir être pris en main rapidement.

3.1.3 À quoi peut servir un modèle ?

Comme nous l'avons expliqué, un modèle est principalement destiné à être utilisé par les êtres humains pour représenter une entité complexe du monde réel, et afin de pouvoir échanger et raisonner sur une représentation simplifiée mais précise de cette entité. Nous pouvons distinguer deux activités principales dans lesquelles l'utilisation d'un modèle est particulièrement efficace.

Premièrement, un modèle peut être utilisé afin d'étudier, d'analyser et de comprendre une situation ou un phénomène réel existant. C'est en particulier une méthode

couramment utilisée dans les domaines de la physique, de la chimie, mais également dans les sciences économiques et sociales. En effet, les chercheurs tentent de modéliser différents phénomènes à l'aide d'équations mathématiques, ou encore de courbes graphiques. De même, les plans et les cartes géographiques constituent des modèles qui permettent de représenter graphiquement et de manière simplifiée une zone géographique. Dans le domaine de l'informatique, il est possible de modéliser le comportement d'un système existant à l'aide, par exemple, de mécanismes de surveillance des performances de ce système. D'autres études récentes proposent de maintenir à l'exécution un ou plusieurs modèles d'un système complexe et dynamique, afin de pouvoir étudier en temps réel différentes caractéristiques de ce système, comme son architecture répartie par exemple, mais également de pouvoir éditer ces modèles pour modifier le comportement du système [Götz 2014].

Deuxièmement, un modèle peut être utilisé dans le but de concevoir une entité qui n'existe pas encore. Quel que soit le domaine d'application, une étape de modélisation peut être nécessaire afin de réfléchir, entre être humains, à ce que l'on cherche à concevoir. Par ailleurs, la définition d'un modèle permet de vérifier certaines propriétés d'entité que l'on conçoit. C'est le cas, par exemple, en architecture ou en mécanique où il est nécessaire de concevoir des plans détaillés de la construction que l'on étudie, afin d'en vérifier la solidité, la résistance aux intempéries ou aux chocs, etc. Dans ce cas, un modèle peut être utilisé comme support de simulation. De même, un modèle peut être utilisé dans le cadre de la mise en place d'un nouveau processus métier afin d'en vérifier l'efficacité, le coût ou le temps d'exécution. Dans le domaine de l'informatique, la modélisation est couramment utilisée et conseillée dans toutes les étapes de conception d'un système informatique, depuis la spécification fonctionnelle du système, jusqu'à la description algorithmique de ces composants. Nous traiterons ce point plus en détail dans la section 3.2.1. Les modèles sont également utilisés à l'exécution pour la réalisation de systèmes réflexifs et reconfigurables [Götz 2014].

3.2 De l'outil de raisonnement à l'outil de production

Dans l'ensemble des exemples présentés précédemment, les modèles sont utilisés afin de réfléchir, d'étudier, de raisonner, ou encore d'élaborer et démontrer des propriétés d'entités réelles existantes ou que l'on cherche à concevoir. Cependant, les modèles

peuvent également être utilisés comme des moyens de production, et c'est en particulier ce que propose l'IDM dans le cadre de l'informatique. Dans cette section nous allons tout d'abord classifier de façon rigoureuse les différents niveaux de modélisation existants en informatique. Cette étape est nécessaire afin de mettre en place les processus de production dirigés par les modèles. Puis nous allons décrire ces processus en détaillant les deux méthodes de production que sont les transformations de modèles et la génération de code. Enfin nous présenterons les objectifs principaux et les applications de l'IDM. L'ensemble de cette section s'appuie principalement sur les travaux de Jézéquel et al. [Jézéquel 2012], Blanc [Blanc 2005], et Bézivin [Bézivin 2001].

3.2.1 Niveaux de modélisation

Afin de pouvoir exploiter les modèles de façon productive, l'IDM propose des compléments aux notions basiques de modèle, et de langage de modélisation. En particulier l'IDM définit une hiérarchie des niveaux de modélisation qui permet de formaliser la notion de langage de modélisation et la définition d'un tel langage. Cette hiérarchie est décrite dans la section 3.2.1.1. De plus l'IDM propose un processus de conception des systèmes informatiques à travers plusieurs étapes de modélisation, depuis la définition des exigences métier jusqu'à l'implémentation, chaque étape donnant lieu à la définition d'un modèle spécialisé. Ce processus ainsi que les types de modèles qui le composent sont décrits dans la section 3.2.1.2.

3.2.1.1 Des concepts au monde réel

Comme nous l'avons vu dans la section 3.1.2, un modèle est défini à partir d'un langage de modélisation. Cependant, pour qu'un modèle puisse être traité et exploité par un ordinateur, il est nécessaire que le langage de modélisation soit lui aussi formellement défini, c'est-à-dire modélisé. Pour cela, l'IDM introduit le concept de métamodèle, qui est un modèle qui sert à définir un modèle. De la même manière, un métamodèle est défini par un métamétamodèle qui, lui, se définit lui-même. Ces trois couches de modélisation, auxquelles s'ajoute la couche représentant le monde réel, sont définies par l'OMG [OMG 2002] [1] [Bézivin 2001], et représentées dans la figure 3.1. Les modèles de

1. La nouvelle version de la spécification MOF de l'OMG [OMG 2014] relâche la contrainte sur le nombre de couches de modélisation.

niveaux M3 et M2 sont des modèles servant à définir d'autres modèles, ils peuvent donc être vus comme des langages de modélisation. Concernant le niveau M3, l'OMG préconise l'utilisation du MOF (*MetaObject Facility*). D'autres langages plus simples sont utilisés dans l'industrie, comme par exemple Ecore. Le niveau M2 est composé de métamodèles, le plus utilisé étant UML. Cependant d'autres métamodèles peuvent être définis dans le but, par exemple, de concevoir des modèles spécifiques à un domaine particulier. Enfin le niveau M1 est composé de modèles tels que les diagrammes UML (diagrammes de classes, de séquences, de cas d'utilisation, etc.) ou de modèles spécifiques définis à l'aide d'autres métamodèles.

Cette hiérarchie permet à tout modèle d'être exploité par l'outil informatique, en raisonnant à partir de son niveau méta. En effet tout raisonnement ou algorithme défini à partir de concepts d'un niveau de modélisation M sera applicable à chaque modèle instance de niveau M-1 de ce modèle. Ce mécanisme est détaillé dans la section 3.2.2.

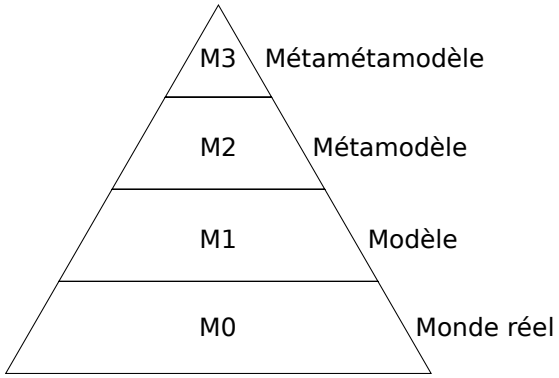

FIGURE 3.1 – Les quatre couches de modélisation de l'IDM

3.2.1.2 Des exigences à l'implémentation

En complément de la hiérarchie présentée dans la section précédente, l'OMG définit différents types de modèles qui correspondent à quatre étapes de conception d'un système informatique. Ces étapes, ainsi que les modèles correspondants, sont représentés dans la figure 3.2. Chacun de ces modèles est de niveau M1, il est donc défini comme une instance d'un métamodèle de niveau M2.

La première étape consiste à décrire un CIM (*Computation Independent Model*). Les CIM sont des modèles qui représentent les fonctionnalités d'un système. Ils doivent

se contenter de décrire à quoi sert le système sans faire intervenir de concepts techniques liés à la conception du système. Un exemple de CIM peut être un diagramme de cas d'utilisation UML. La deuxième étape consiste à définir un PIM (*Platform Independent Model*). Ce modèle est plus technique car il décrit l'architecture logicielle du système, mais sans prendre en compte les spécificités d'une plateforme particulière. Un PIM peut, par exemple, décrire les différents composants, modules, ou services d'un système (p. ex., diagramme de classes UML). La troisième étape est la définition d'un PSM (*Platform Specific Model*), c'est-à-dire une extension du PIM avec des concepts spécifiques à une plateforme cible (p. ex., JEE [2]). Une plateforme peut elle-même être modélisée par un PDM (*Platform Description Model*). Enfin, la quatrième et dernière étape est la définition du code qui peut également être vu comme un modèle.

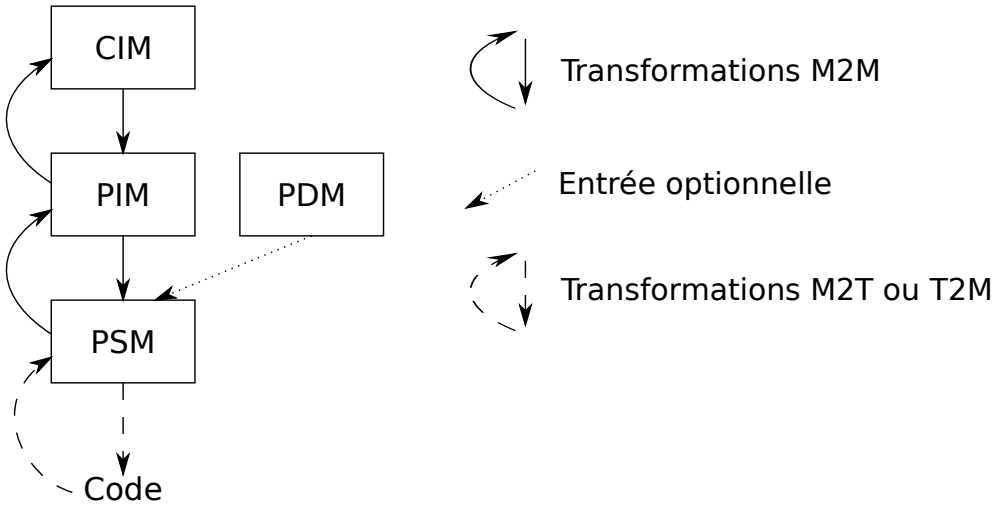

FIGURE 3.2 – Étapes de conception de l'IDM

L'OMG précise qu'il est nécessaire de maintenir des liens de traçabilité entre les différentes étapes de modélisation. Par exemple, il est nécessaire de maintenir un lien entre une fonctionnalité décrite dans un CIM et les éléments des PIM et PSM qui l'implémentent. Ces liens de traçabilité, couplés aux méthodes de transformation et de génération présentées section 3.2.2 garantissent l'efficacité de la chaîne de conception d'un système par une approche IDM.

2. http://www.oracle.com/technetwork/java/javaee/overview/index.html

3.2.2 Des modèles productifs

En plus d'accompagner la conception de systèmes informatiques, en définissant différentes étapes de modélisation, l'IDM propose de rendre les modèles productifs, en automatisant une partie de la chaîne de conception d'un système. Ces moyens de production peuvent être résumés en deux types de transformation : les transformations M2M (*Model To Model*), ou transformations de modèles, et les transformations M2T (*Model To Text*), ou générations de code.

3.2.2.1 Transformation de modèles

La première méthode est la transformation de modèles, M2M, illustrée dans la figure 3.3. Cette méthode consiste à écrire des règles de transformation d'un modèle en entrée Ma vers un modèle de sortie Mb (niveau M1). La transformation Tab (niveau M1) manipule les concepts du niveau méta (ou M2) des modèles en entrée et en sortie, respectivement MMa et MMb sur la figure. De plus, une transformation est elle-même modélisée, ou décrite, par un langage (ou métamodèle) de transformation de modèle MMt (niveau M2). Enfin, comment nous l'avons vu précédemment, les métamodèles MMa, MMb et MMt sont des instances d'un métamétamodèle (niveau M3) MMM.

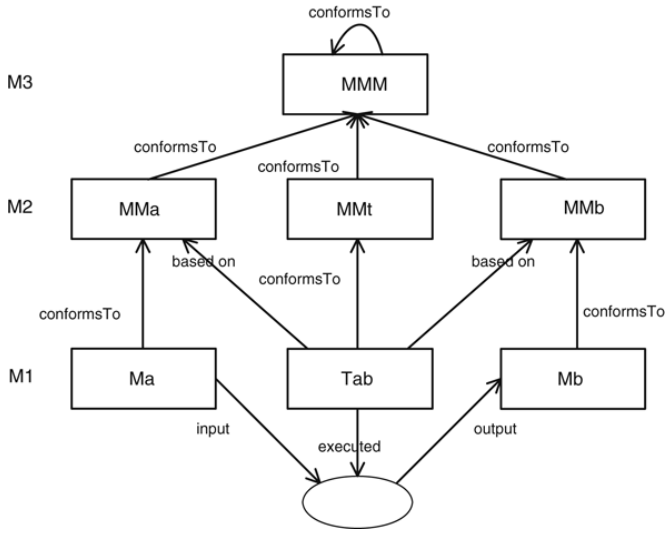

FIGURE 3.3 – Transformation de modèles [Jouault 2008]

La transformation de modèle fait l'objet d'un standard proposé par l'OMG ap-

pelé QVT [OMG 2011]. Il existe également plusieurs *frameworks* de transformation de modèles, comme par exemple ATL [3] [Jouault 2008].

Cette méthode de transformation M2M peut, par exemple, être utilisée pour automatiser la définition des PSMs à partir d'un PIM. Une telle transformation ferait correspondre les concepts de composants, ou de modules, aux concepts spécifiques à une plateforme comme les EJBs pour une plateforme JEE. Ces derniers pouvant être définis dans le PDM de la plateforme JEE.

3.2.2.2 Génération de code

La deuxième méthode est la transformation M2T qui consiste en la génération de code à partir d'un modèle. Là encore, les règles de génération sont écrites à partir des concepts de niveau méta du modèle en entrée. De même que pour les transformations M2M, il existe plusieurs langages qui implémentent les spécifications M2T, comme le langage de *template* Acceleo [4].

La génération de code est l'étape qui permet l'automatisation du passage d'un PSM au code. Cette génération peut soit générer un code complet, directement exécutable, soit un code qui doit être complété par un programmeur. Un exemple de génération de code peut être la génération de classes Java correspondant aux classes d'un diagramme de classes UML.

3.2.3 Objectifs et applications

Comme nous l'avons vu dans les sections précédentes, l'ensemble des propositions de l'IDM a pour but de mettre les modèles au centre de la chaîne de conception d'un système informatique. Cette approche présente plusieurs avantages. D'abord, elle permet d'assurer la pérennité et la maintenabilité d'un système. En effet, les liens de traçabilité entre les différents modèles de conception d'un système permettent de s'assurer qu'un changement dans l'un de ces modèles sera répercuté dans les autres. Par exemple, la modification d'une exigence d'un CIM entraînera la modification du PIM et du PSM correspondants, et donc la re-génération du code. À l'inverse, afin de respecter les liens

3. http://www.eclipse.org/atl/
4. http://www.eclipse.org/acceleo/

de traçabilité, toute modification du code ayant un impact sur l'architecture logicielle ou sur les fonctionnalités du système devra entraîner la modification des modèles PSM, PIM et CIM correspondants. Cette démarche permet donc d'assurer la validité des modèles au cours de l'évolution du système, ce qui permet de continuer à utiliser ces modèles comme moyens de réflexion ou de communication entre être humains tout au long de la vie du système.

Ensuite, la possibilité de définir ses propres métamodèles permet de définir des modèles spécifiques au domaine métier dans lequel on travaille. Cela permet d'adapter la chaîne de conception aux besoins spécifiques de chaque système, et de rendre l'élaboration de ces modèles accessible aux experts du domaine. Ceci est particulièrement utile pour définir des CIMs représentant des exigences métiers, voire des PIMs décrivant des architectures logicielles spécifiques.

Enfin, l'approche IDM permet d'automatiser plusieurs étapes de la chaîne de conception d'un système, ce qui permet de produire des systèmes de façon plus rapide, et donc à moindre coût.

3.3 L'IDM pour l'organisation des systèmes répartis complexes

L'IDM est d'autant plus utile pour la conception des systèmes complexes. En effet, différents niveaux d'abstraction et différentes perspectives s'avèrent nécessaires pour concevoir de tels systèmes [Schmidt 2006], [France 2007]. Les auteurs de [France 2007] distinguent deux classes de modèles :

Les modèles utilisés à la conception Un système complexe peut être décrit à l'aide de modèles à différents niveaux d'abstraction, selon différents points de vue ou préoccupations (par exemple, architecture ou modèle de déploiement). Ces modèles peuvent être utilisés pour générer le code applicatif par transformations. L'utilisation de cette démarche permet de garantir, d'une part, l'indépendance de la modélisation par rapport aux plateformes utilisées (il est possible de changer de technologie sans remettre en cause la modélisation), et d'autre part, l'évolutivité du système (l'enrichissement du système est considéré au niveau des

modèles et le système exécutable est regénéré). Ces points sont également soulignés dans [Schmidt 2006]. Dans cet article, l'auteur insiste sur la nécessité de l'indépendance des systèmes par rapport aux plateformes, dans un contexte où ces dernières sont en constante évolution, et où de nouvelles plateformes apparaissent régulièrement. Selon l'auteur, la portabilité « manuelle » du code pour suivre ces évolution est irréaliste. Cette approche est d'autant plus nécessaire pour un système complexe réparti sur plusieurs niveaux de l'infrastructure ;

Les modèles utilisés à l'exécution [Blair 2009] Ces modèles présentent des vues du système en cours d'exécution suivant certains aspects. Ils sont donc des abstractions de l'exécution. L'utilisation de tels modèles vise à favoriser l'adaptation du système à un contexte d'exécution dynamique.

En ce qui concerne l'IDM pour les systèmes complexes, de nombreux travaux ont été réalisés. Nous citons parmi eux : l'IDM pour la conception et l'exécution des systèmes pervasifs sensibles au contexte [Serral 2010], [Chabridon 2013], les modèles à l'exécution pour l'adaptation des systèmes [Morin 2009], les modèles à l'exécution pour la gestion de systèmes *multi-cloud* [Ferry 2013], l'adéquation de plusieurs *frameworks* de modèles à l'exécution sur des infrastructures variées allant des objets communicants aux plateformes de *clouds* [Fouquet 2014].

Notre approche consiste à proposer une vision multiéchelle d'un système complexe. Cependant, nos travaux se distinguent de la vision multiéchelle proposée par [Gassara 2013]. Dans cet article les auteurs proposent un profil UML multiéchelle pour le déploiement de systèmes complexes. Le profil présenté propose de considérer trois « échelles » fixes : l'infrastructure, la communication, et les entités déployées. Dans leur proposition, les échelles correspondent à des vues du système complexe. Il n'y a pas de distinction de niveaux dans une vue donnée. Nous proposons de définir une vision particulière pour considérer l'aspect multiéchelle d'un système réparti complexe. Notre proposition vise à permettre d'identifier la forte hétérogénéité de ces systèmes, dans le but d'aider les concepteurs à en appréhender la complexité.

Deuxième partie

Contribution – MuSCa : framework de caractérisation multiéchelle

Chapitre 4

Processus de caractérisation multiéchelle

Sommaire

Dans ce chapitre, nous présentons le processus de caractérisation multiéchelle qui permet de proposer une vision simplifiée d'un système réparti complexe. Dans la section 4.1, nous proposons un vocabulaire multiéchelle, nous envisageons l'étude d'un système complexe suivant plusieurs points de vue (par exemple, équipement et géographie), enfin nous énonçons le processus de caractérisation multiéchelle. Puis, dans la section 4.2, nous proposons une formalisation du vocabulaire multiéchelle, et en particulier du concept d'échelle. Cette formalisation s'appuie sur des concepts mathématiques de la théorie des ensembles. Enfin nous terminons par une conclusion de ce chapitre en section 4.3.

4.1 Concepts et processus de caractérisation multiéchelle

La définition d'un vocabulaire de caractérisation multiéchelle permet de partager un ensemble de concepts organisés. L'utilisation de ces concepts permet de construire une vision simplifiée des systèmes répartis multiéchelles.

Dans cette section, nous commençons par définir, dans la section 4.1.1, le vocabulaire multiéchelle proposé. Puis, dans la section 4.1.2, nous illustrons l'étude de la nature multiéchelle d'un système réparti complexe suivant différents points de vue identifiés. Nous continuons, dans la section 4.1.3, en soulignant l'existence de plusieurs types de dimensions. Enfin, nous énonçons le processus de caractérisation multiéchelle d'un système réparti complexe en section 4.1.4.

4.1.1 Vocabulaire multiéchelle

L'analyse multiéchelle d'un système réparti complexe proposée dans cette thèse repose sur les concepts de point de vue, dimension, mesure, échelle et ensemble d'échelles, introduits ci-dessous.

Point de vue En général, l'architecture d'un système est obtenue en étudiant ce système depuis différents *points de vue*. L'observation d'un système depuis un certain point de vue forme une *vue* du système [ISO/IEC/IEEE 2011]. Une vue est constituée de *projections* des entités du système sur ce point de vue.

Dimension Chaque point de vue est étudié à travers différentes *dimensions*. Une dimension multiéchelle est une caractéristique mesurable d'une vue d'un système depuis un certain point de vue.

Mesure Une dimension est associée à une *mesure*, numérique ou sémantique. Dans le cas d'une mesure sémantique, les valeurs de cette mesure doivent être comparables entre elles et ordonnées.

Échelle À l'aide d'une mesure, une dimension peut être divisée en *échelles*. Une échelle correspond à un intervalle de valeurs de cette mesure. Les entités d'un système associées à une même échelle sont considérées comme homogènes pour le point de vue et la dimension considérés. Ainsi, afin de réduire la complexité présente dans un système, en terme d'hétérogénéité, les échelles doivent être

suffisamment distinctes les unes des autres. Par exemple, pour une mesure numérique, les échelles peuvent correspondre à des intervalles dont les centres sont distants de plusieurs ordres de grandeurs.

Ensemble d'échelles Un *ensemble d'échelles* contient les échelles choisies pour un couple dimension/mesure donné.

Remarque. *Les termes* mesure*,* échelle*, et* ensemble d'échelles *sont définis de manière plus formelle à l'aide de définition ensemblistes dans la section 4.2.*

L'identification de plusieurs échelles pour un point de vue donné, dans une dimension donnée, avec une mesure donnée, permet de mettre en évidence la présence d'hétérogénéité dans le système réparti multiéchelle.

4.1.2 Étude de points de vue multiéchelles

Afin d'illustrer le processus de caractérisation multiéchelle, nous présentons dans cette section l'étude de plusieurs systèmes répartis multiéchelles depuis les points de vue suivants : équipement, donnée, réseau, géographie et utilisateur. Ces différents points de vue sont couramment identifiés dans l'étude des systèmes répartis complexes, comme par exemple dans les modèles de contexte [Dey 2001]. Pour chacun des points de vue, nous commençons par présenter un système qui met en avant le point de vue étudié. Pour ce système, nous sélectionnons une ou plusieurs dimensions, et pour chaque dimension nous proposons un ensemble d'échelles pertinentes pour la caractérisation multiéchelle du système étudié. Ainsi, nous illustrons que chaque système peut donner lieu à une caractérisation multiéchelle particulière.

4.1.2.1 Point de vue équipement

En étudiant un système depuis le point de vue équipement, nous considérons les équipements qui le composent et faisons abstraction des autres composantes du système. Les auteurs de [Blair 2012] présentent un scénario de surveillance de l'environnement, et plus particulièrement de surveillance de crue. Ce scénario fait intervenir une grande variété d'équipements. Des capteurs sont utilisés pour surveiller une rivière.

Les données obtenues par les capteurs sont envoyées vers le *cloud* car le traitement de ces informations requiert une puissance de calcul très élevée. Enfin, les informations scientifiques et les alertes de sécurité sont accessibles depuis les équipements mobiles des différents acteurs du système (scientifiques, services d'urgence, tout public).

Nous avons décidé d'étudier la nature multiéchelle de ce scénario, depuis le point de vue équipement, à travers deux dimensions, associées respectivement à deux mesures numériques. La première dimension est la capacité de stockage (en octets). La seconde est la puissance de calcul (en instructions par seconde). Dans la figure 4.1, nous positionnons certaines familles d'équipements par rapport à leur dispersion estimée pour ces deux mesures (capacité de stockage en haut et puissance de calcul en bas). Pour chaque dimension, nous choisissons plusieurs échelles pertinentes. Pour la dimension capacité de stockage, nous choisissions trois échelles : les échelles *kilo octets* en-dessous de $10^6 o$, *giga octets* entre $10^6 o$ et $10^{12} o$, et *peta octets* au-dessus de $10^{12} o$. Pour la dimension puissance de calcul, nous choisissons les échelles *kilo IPS* en-dessous de $10^6 IPS$, *giga IPS* entre $10^6 IPS$ et $10^{12} IPS$, et *peta IPS* au-dessus de $10^{12} IPS$. Chaque mesure est présentée avec une échelle logarithmique. Ainsi, nous pouvons remarquer que les centres de deux échelles contiguës sont distants de plusieurs ordres de grandeurs.

Ces échelles représentent les ruptures de technologies existantes entre les équipements d'un système. Ainsi, deux équipements associés respectivement à deux échelles différentes, pour une dimension donnée, peuvent être considérés comme hétérogènes pour cette dimension, ce qui peut impliquer des fonctions différentes au sein du système. Par exemple, des équipements associés à des échelles différentes pour la dimension capacité de stockage peuvent être utilisés pour stocker ou manipuler des données de volumes significativement différents. De même, pour la dimension puissance de calcul, les différentes échelles peuvent être associées à différents types de calculs, qui requièrent plus ou moins de puissance.

À travers les deux dimensions étudiées, nous pouvons mettre en lumière la nature multiéchelle du système réparti de surveillance de crue depuis le point de vue équipement. En effet, pour ces deux dimensions, les équipements du système sont associés à plusieurs échelles. Par exemple, les capteurs, les équipements mobiles et les *clouds* sont associés respectivement aux échelles *kilo octets*, *giga octets* et *peta octets* pour la dimension capacité de stockage.

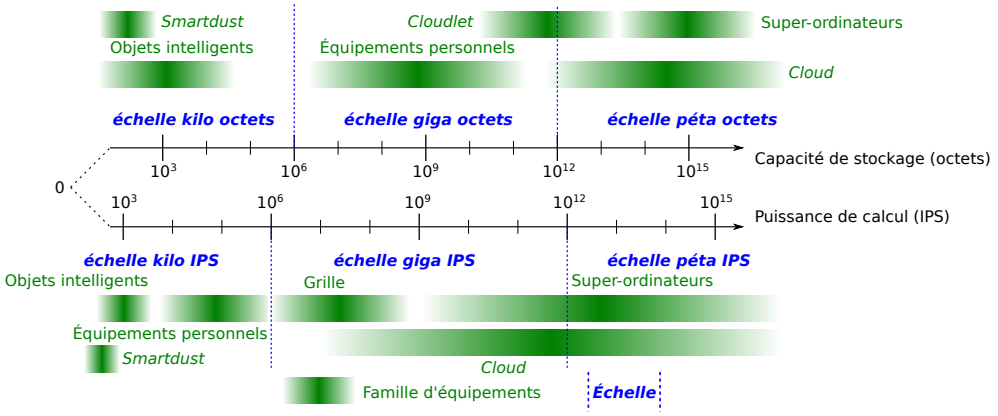

FIGURE 4.1 – Dimensions et échelles pour le point de vue équipement

4.1.2.2 Point de vue donnée

Le scénario de surveillance de crue [Blair 2012] peut également être étudié suivant le point de vue donnée. Pour le point de vue donnée, nous considérons les données manipulées par le système. Les données provenant des capteurs sont transformées puis stockées dans le *cloud* afin de maintenir un historique des données et d'être disponibles pour des analyses statistiques. Dans le *cloud*, les données sont consolidées (c'est-à-dire analysées, structurées et synthétisées). Dans le scénario d'analyse de crues comme dans d'autres scénarios, dans le cadre de l'IoT, le volume des données présentes varie grandement depuis les données atomiques produites par des capteurs individuels jusqu'aux données *big data* obtenues par le rassemblement d'un grand nombre de capteurs, potentiellement répartis dans le monde entier, et stockées dans le *cloud*.

Comme le montre la figure 4.2, pour le point de vue donnée, nous avons choisi la dimension volume de données, mesurée en octets. Pour cette dimension nous identifions deux échelles pertinentes pour le scénario de surveillance de crue : l'échelle *très petit volume*, en-dessous de $10^4 o$, pour les données de capteur, et l'échelle *très gros volume*, au-dessus de $10^{14} o$, pour l'historique des données. Entre ces deux échelles extrêmes, explicites dans le scénario de surveillance de crue, nous pouvons identifier deux autres échelles qui nous semblent pertinentes pour d'autres systèmes : l'échelle *petit volume*, entre $10^4 o$ et $10^7 o$, et l'échelle *gros volume*, entre $10^7 o$ et $10^{14} o$.

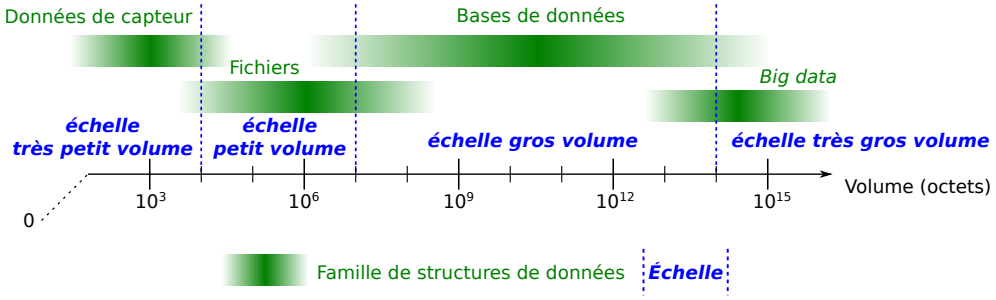

FIGURE 4.2 – Dimensions et échelles pour le point de vue donnée

4.1.2.3 Point de vue réseau

Pour étudier un système depuis le point de vue réseau, nous considérons les réseaux traversés par le système. Les auteurs de [Kessis 2009] proposent un middleware pour la gestion de ressources en contexte multiéchelle. Le système proposé a pour but de permettre une gestion flexible de ressources. Le terme multiéchelle est utilisé pour décrire la répartition des ressources sur différents réseaux de différentes portées : des réseaux personnels (*Personal Area Network*, PAN), des réseaux locaux (*Local Area Network*, LAN), et des réseaux étendus (*Wide Area Network*, WAN).

Nous illustrons la nature multiéchelle du point de vue réseau à travers trois dimensions, associées respectivement à trois mesures numériques. La figure 4.3 illustre ces trois dimensions : la dimension latence mesurée en millisecondes, la dimension bande passante mesurée en kilobits par seconde et la dimension portée mesurée en mètres. Puis nous positionnons certaines familles de réseaux (par exemple, zigbee, WiFi, satellite) par rapport à la répartition estimée de ces familles pour les trois mesures considérées. Ensuite, pour chaque couple dimension/mesure, nous identifions des échelles pertinentes. Pour la latence, nous identifions deux échelles : l'échelle de *faible latence* en-dessous de $1s$ et l'échelle de *forte latence* au-dessus de $1s$. Pour la bande passante, nous identifions trois échelles : l'échelle de *faible bande passante* en-dessous de $10^3 kb/s$, l'échelle de *moyenne bande passante* entre 10^3 et $10^6 kb/s$, et l'échelle de *forte bande passante* au-dessus de $10^6 kb/s$. Enfin, pour la portée, nous identifions les échelles PAN, LAN, MAN (Metropolitan Area Network), et WAN.

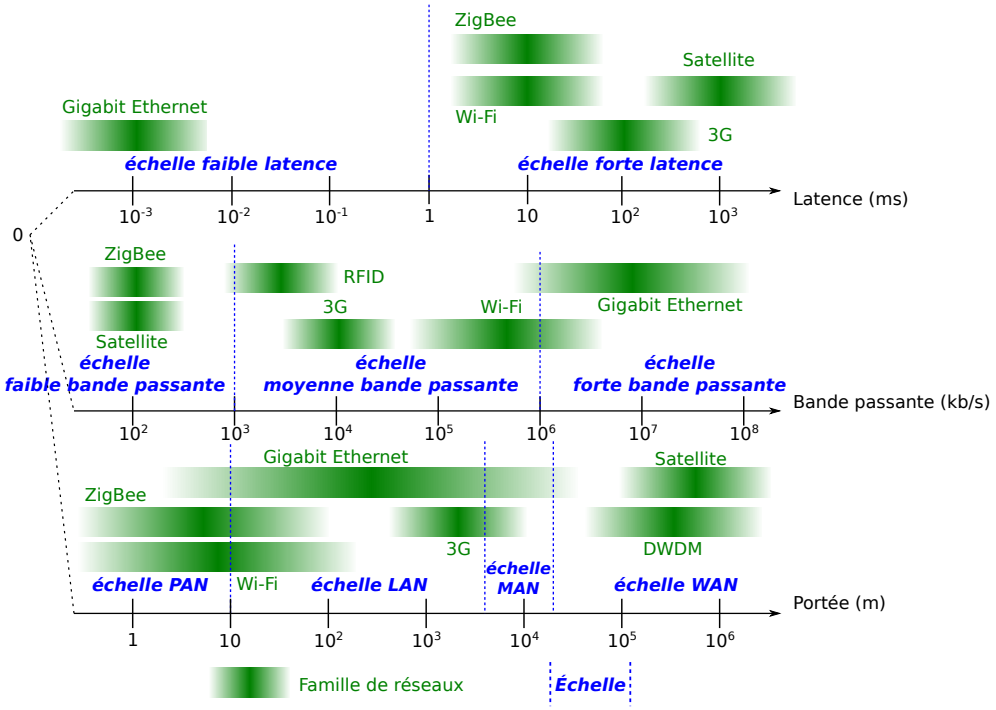

FIGURE 4.3 – Dimensions et échelles pour le point de vue réseau

4.1.2.4 Point de vue géographie

Le point de vue géographie est pertinent pour étudier la nature multiéchelle d'un système réparti complexe. Pour ce point de vue, nous considérons les coordonnées GPS des entités du système. Les auteurs de [Franklin 2005] présentent une base de données hautement répartie pour l'IoT. Dans ce système, le point de vue géographie est utilisé pour définir différentes zones de différentes échelles afin de construire un système hiérarchique. Pour chaque zone, un ordinateur dédié est responsable du traitement des requêtes et de la gestion des données : par exemple, pour les zones ambiantes, régionales, ou pour le monde entier.

Nous illustrons le point de vue géographie à travers deux dimensions, comme le montre la figure 4.4, l'une associée à une mesure numérique et l'autre à une mesure sémantique. La première dimension est la dimension distance qui mesure en mètres la distance maximale entre deux éléments d'un ensemble d'entités. Pour cette mesure numérique, nous avons choisi un ensemble de quatre échelles : l'échelle *locale* en-dessous de $10m$, l'échelle *accessible à pied* entre $10m$ et 10^3m, l'échelle *accessible en voiture*

entre 10^3m et 10^5m, et l'échelle *accessible en avion* au-dessus de 10^5m. La deuxième dimension est la dimension localisation administrative, qui est associée à une mesure sémantique. Elle s'applique également à un ensemble d'entités et mesure la plus petite localisation commune à cet ensemble d'entités. Nous avons retenu un ensemble de six échelles : l'*immeuble*, le *quartier*, la *ville*, la *région*, le *pays*, et le *monde*. L'organisation d'un système et les interactions entre les entités de ce système peuvent varier en fonction de la dimension et des échelles choisies dans le point de vue géographie. D'autre part, comme le suggère le scénario présenté section 1.4, les échelles définies pour la dimension localisation administrative peuvent être utilisées pour spécifier qu'un certain composant doit être installé dans chaque ville, ou dans chaque quartier.

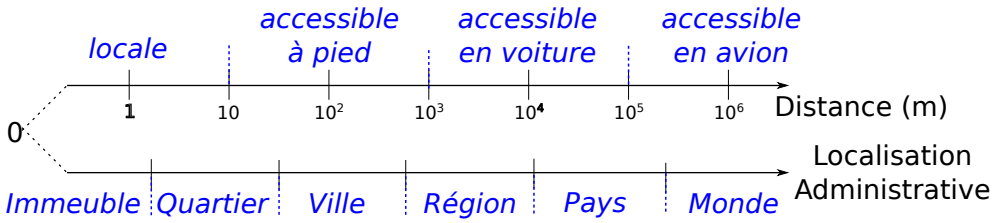

FIGURE 4.4 – Dimensions et échelles pour le point de vue géographie

4.1.2.5 Point de vue utilisateur

Le dernier point de vue présenté est le point de vue utilisateur. Pour ce point de vue, nous considérons des ensembles d'utilisateurs qui accèdent au système. Pour illustrer ce point de vue, nous prenons pour exemple le système de gestion de contexte proposé par le projet ANR INCOME [Arcangeli 2012]. Dans ce système, pour des raisons de protection de la vie privée, il est nécessaire de prendre en compte différentes catégories d'utilisateurs et d'ensembles d'utilisateurs. En effet, une information a des propriétés de confidentialité différentes en fonction des catégories d'utilisateurs auxquels elle est présentée. Le système considère différemment un utilisateur isolé, les groupes, les communautés, et le reste du monde.

Comme le montre la figure 4.5, pour ce système, nous avons choisi d'étudier le point de vue utilisateur à travers deux dimensions. La première dimension est mesurée par le nombre d'utilisateurs qui composent un ensemble d'utilisateurs du système. Pour cette dimension nous avons identifié quatre échelles : l'*individu* pour 1 utilisateur, le *groupe* entre 2 et 10^2, la *communauté* entre 10^2 et 10^5, et *tous* au-dessus de 10^5.

La deuxième dimension est associée à une mesure sémantique : nous considérons la politique d'accès utilisée quand un nouvel utilisateur souhaite joindre l'ensemble d'utilisateurs. Nous considérons plusieurs politiques d'accès : impossible (l'ensemble d'utilisateurs est fermé), contrôlé (il est possible de joindre l'ensemble avec une autorisation), ouvert (l'ensemble peut être rejoint part une simple inscription), et public (aucune inscription n'est nécessaire). Chaque politique d'accès est associée à une échelle : *individu*, *groupe*, *communauté*, et *tous*.

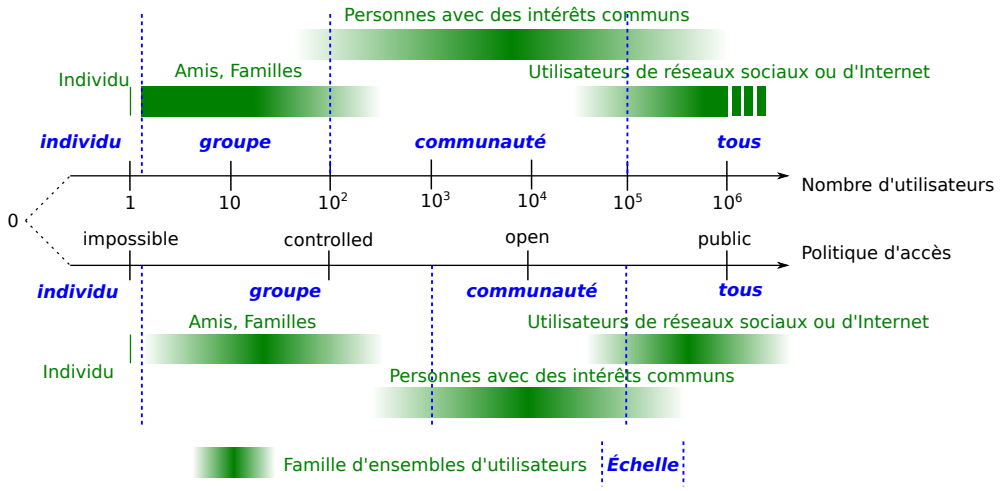

FIGURE 4.5 – Dimensions et échelles pour le point de vue utilisateur

4.1.3 Typologie des dimensions multiéchelles

À partir des exemples précédents, nous distinguons les dimensions multiéchelles hiérarchiques et les non hiérarchiques.

Par exemple, si nous considérons la dimension localisation administrative, pour le point de vue géographie, et la dimension capacité de stockage, pour le point de vue équipement, nous remarquons que les échelles pertinentes construites pour ces dimensions ne sont pas organisées de la même manière. En effet, pour la dimension localisation administrative (cf. figure 4.4), les échelles sont organisées les unes *dans* les autres, c'est-à-dire de manière hiérarchique : un quartier *est dans* une ville, une ville *est dans* une région, etc. À l'inverse, les échelles de la dimension capacité de stockage (cf. figure 4.1) sont organisées les unes *à côté* des autres, c'est-à-dire de manière non

hiérarchique : on ne peut pas dire qu'un équipement de l'échelle kilo octets *est dans* un équipement de l'échelle giga octets.

Cette distinction entre dimensions hiérarchiques et non hiérarchiques nous amène à envisager deux façons de caractériser les entités d'un système de façon multiéchelle. La méthode la plus intuitive consiste à mesurer chaque entité du système de façon indépendante. Par exemple, pour le point de vue équipement et pour l'ensemble d'échelles choisi dans la figure 4.1, chaque équipement du système peut être associé à une échelle en fonction de sa capacité de stockage. Notons qu'il est possible qu'une entité soit associée à plusieurs échelles si les échelles de l'ensemble d'échelles se chevauchent (ce point est détaillé dans la section 4.2.2). Cependant, dans le cas d'une dimension hiérarchique, les échelles étant organisées les unes *dans* les autres, une entité du système peut être associée à toutes les échelles d'un ensemble d'échelles. Par exemple, pour la dimension localisation administrative du point de vue géographie, et pour l'ensemble d'échelles choisi dans la figure 4.4, toute entité du système qui peut être associée à un immeuble est également associée à un quartier, une ville, une région, etc. C'est pourquoi, dans le cas d'une dimension hiérarchique, nous proposons des mesures appliquées à un ensemble d'entités. Par exemple, si l'on considère trois entités d'un système, l'une située à Toulouse, et les deux autres à Paris, nous pouvons associer l'ensemble de ces trois entités à l'échelle pays, qui est la plus petite échelle commune à ces trois entités.

Pour les dimensions hiérarchiques, nous favorisons les mesures appliquées à un ensemble d'entités. Nous pouvons ainsi mettre en évidence des interactions entre entités à plusieurs niveaux d'échelles (quartier, ville, pays). Pour les dimensions non hiérarchiques nous appliquerons les mesures à une entité.

4.1.4 Énoncé du processus de caractérisation multiéchelle

L'étude précédente nous permet d'énoncer le processus de caractérisation multiéchelle.

Processus de caractérisation multiéchelle. *La nature multiéchelle d'un système réparti complexe doit être étudiée indépendamment depuis chacun des points de vue choisis par le concepteur du système. Pour chaque point de vue, une vue restreinte du système est considérée. Puis, pour cette vue, une ou plusieurs dimensions sont choisies. Afin d'identifier les échelles des dimensions étudiées, chaque dimension est associée*

à une mesure numérique ou sémantique. Les échelles sont des intervalles de valeurs de cette mesure. Le choix des points de vue, des dimensions/mesures, et des échelles pertinentes pour une caractérisation multiéchelle reste ouvert. C'est au concepteur du système de choisir en fonction des propriétés du système qu'il veut étudier. Ces choix lui permettent de mettre en évidence des caractéristiques spécifiques à gérer au cours des phases de conception et/ou d'exécution du système.

Nature multiéchelle d'un système. *La nature multiéchelle d'un système est relative à une caractérisation multiéchelle. Pour un point de vue donné, et pour un couple dimension/mesure, chaque projection d'entité —ou ensemble de projections d'entités, dans le cas d'une dimension hiérarchique (cf. section 4.1.3)— d'une vue restreinte du système est associée à une échelle. Pour une caractérisation donnée, et un point de vue donné, un système distribué est qualifié de multiéchelle quand, pour au moins une dimension, les projections d'entités —ou ensembles de projections d'entités— de sa vue sont associées à des échelles différentes. L'identification de la nature multiéchelle d'un système met en évidence la présence d'hétérogénéité (technique ou non technique) pour ce point de vue, cette dimension et cette mesure.*

4.2 Formalisation ensembliste de la caractérisation multiéchelle

Afin de préciser le vocabulaire multiéchelle, et d'accompagner le concepteur de systèmes dans la caractérisation d'un système réparti complexe, nous avons identifié plusieurs propriétés que doit respecter une caractérisation multiéchelle. Nous nous appuyons pour cela sur des concepts de la théorie des ensembles [Bourbaki 1970]. Dans cette section nous raisonnons au niveau d'un ensemble d'échelles, pour un point de vue et un couple dimension/mesure donnés. Une entité du système est représentée par sa projection dans le point de vue étudié. Nous considérons une échelle comme un intervalle de valeurs pour la dimension/mesure étudiée. Une entité —ou un ensemble d'entités— appartient à une échelle si la valeur mesurée de sa projection pour la dimension étudiée appartient à cette échelle. Afin de simplifier notre étude nous considérons que, pour un système donné, l'ensemble des valeurs possibles d'une mesure est borné par une borne MIN inférieure à toute les valeurs possibles de la mesure et par une borne MAX strictement supérieure à toutes les valeurs possibles de la mesure. Par

exemple, nous supposons que la capacité de stockage d'un équipement d'un système donné est supérieure à 0 et strictement inférieure à une borne supérieure prédéfinie.

4.2.1 Notations utilisées et association d'un élément à une échelle

Pour un système $\mathcal{S}$, constitué d'entités, pour un point de vue v, une dimension d, et une mesure m donnés, nous utiliserons les notations suivantes :

- $\mathcal{X}_v$ est l'ensemble des projections d'entités de $\mathcal{S}$ sur v,
- $P(\mathcal{X}_v)$ est l'ensemble des parties de $\mathcal{X}_v$, c'est-à-dire l'ensemble des sous-ensembles de $\mathcal{X}_v$,
- $\mathcal{M}$ est l'ensemble, totalement ordonné, des valeurs de la mesure m pour la dimension d pour tout système,
- $\mathcal{M}_\mathcal{S}$ est l'ensemble des valeurs de la mesure m pour un système donné $\mathcal{S}$, il peut s'écrire $\mathcal{M}_\mathcal{S} = [MIN; MAX[|(MIN, MAX) \in \mathcal{M}^2$,
- $\mathcal{E}$ est un ensemble, sur lequel est appliquée la mesure m, égal à $\mathcal{X}_v$ si m s'applique à une entité, ou à $P(\mathcal{X}_v) \setminus \emptyset$ si m s'applique à un ensemble d'entités,
- μ est une fonction de $\mathcal{E}$ dans $\mathcal{M}_\mathcal{S}$, qui associe à une projection —ou à un ensemble de projections— d'entités une valeur mesurée pour d,
- $\mathcal{P}(\mathcal{M}_\mathcal{S})$ est l'ensemble des parties de $\mathcal{M}_\mathcal{S}$,
- $\mathcal{I}$ est l'ensemble des intervalles semi-ouverts à droite de l'ensemble totalement ordonné $\mathcal{M}_\mathcal{S}$, on a $\mathcal{I} \subset \mathcal{P}(\mathcal{M}_\mathcal{S})$,
- $S \in \mathcal{I}$ est une échelle,
- Une échelle S peut donc s'écrire $S = [a; b[|(a, b) \in \mathcal{M}_\mathcal{S}^2$,
- $(S_i)_{i \in I} \subset \mathcal{I}$ est l'ensemble d'échelles choisi pour le couple (d, m) dans le point de vue v.

En plus de ces notations, nous définissons la relation binaire *d'association d'une projection —ou d'un ensemble de projections— d'entités à une échelle* de la manière suivante :

Définition 4.1. *L'association d'une projection —ou d'un ensemble de projections— d'entités e à une échelle S est définie par la relation binaire $\mathcal{A}$ de $\mathcal{E}$ vers $\mathcal{I}$ telle que :*

$$\forall e \in \mathcal{E}, \quad \forall S \in \mathcal{I}, \quad e\mathcal{A}S \iff \mu(e) \in S.$$

4.2.2 Propriétés ensemblistes des ensembles d'échelles d'une caractérisation

Dans cette section nous recommandons plusieurs contraintes à respecter par les ensembles d'échelles d'une caractérisation, afin de garantir certaines bonnes propriétés, comme par exemple de pouvoir associer au moins une échelle à toute entité ou ensemble d'entités du système.

Tout d'abord, nous proposons la contrainte suivante :

Contrainte 4.1. *(C4.1) Un ensemble d'échelles $(S_i)_{i\in I}$, défini pour le couple (d, m), est un recouvrement de $\mathcal{M}_\mathcal{S}$.*

Le terme recouvrement est utilisé dans un sens ensembliste, c'est-à-dire avec la définition suivante :

$$\bigcup_{i\in I} S_i = \mathcal{M}_\mathcal{S}$$

D'après cette définition, la contrainte 4.1 implique que toute valeur mesurable de la dimension étudiée appartient à au moins une échelle de l'ensemble d'échelles, c'est-à-dire :

$$\forall y \in \mathcal{M}_\mathcal{S}, \quad \exists i \in I \quad | \quad y \in S_i.$$

À plus forte raison, cela signifie que nous avons la propriété suivante :

Propriété 4.1. *Pour tout élément e de $\mathcal{E}$ (représentant une projection ou un ensemble de projections d'entités), et pour tout ensemble d'échelles $(S_i)_{i\in I}$ respectant la contrainte 4.1, il existe au moins une échelle de cet ensemble d'échelles à laquelle e est associé :*

$$\forall e \in \mathcal{E}, \quad \forall (S_i)_{i\in I} | C4.1, \quad \exists i \in I \quad | \quad e\mathcal{A}S_i$$

Nous ajoutons la contrainte suivante, qui assure de ne pas définir d'échelle vide :

Contrainte 4.2. *(C4.2) Un ensemble d'échelles $(S_i)_{i\in I}$, défini pour le couple (d, m), doit être composé d'échelles non vides :*

$$\forall i \in I, \quad S_i \neq \emptyset$$

Ces deux contraintes recommandées (contraintes 4.1 et 4.2) impliquent deux cas possibles. Soit l'ensemble d'échelles forme une partition de la dimension, c'est-à-dire un recouvrement ne contenant aucune échelle vide et dans lequel les échelles sont deux à deux disjointes, soit il existe des échelles ayant une intersection non vide.

Dans le premier cas, celui de la partition, nous ajoutons donc la contrainte suivante :

Contrainte 4.3. *(C4.3) Un ensemble d'échelles $(S_i)_{i\in I}$ contient des échelles disjointes deux à deux :*

$$\forall (i,j) \in I^2, \quad S_i \cap S_j = \emptyset$$

Dans ce cas, il n'y a pas d'ambiguïté pour caractériser une entité ou un ensemble d'entités. En effet, la valeur mesurée d'une projection ou d'un ensemble de projection, pour la dimension étudiée, est associée à une et une seule échelle, c'est-à-dire que nous avons la propriété suivante :

Propriété 4.2. *Pour tout élément e de $\mathcal{E}$ (représentant une projection ou un ensemble de projections d'entités), et pour tout ensemble d'échelles $(S_i)_{i\in I}$ respectant les contraintes 4.1, 4.2 et 4.3, il existe une et une seule échelle de cet ensemble d'échelles à laquelle e est associé :*

$$\forall e \in \mathcal{E}, \quad \forall (S_i)_{i\in I} | (C4.1 \wedge C4.2 \wedge C4.3), \quad \exists ! i \in I \quad | \quad e \mathcal{A} S_i$$

Le cas où les échelles constituent une partition de la dimension est un cas idéal, très simple à gérer (pas d'ambiguïté à lever). Mais il peut être simpliste et appauvrir la perception du système multiéchelle.

Dans le deuxième cas, nous n'ajoutons pas la contrainte 4.3, ce qui signifie qu'il est possible qu'une projection d'entité (ou un ensemble de projections) soit associée à plusieurs échelles. Il peut alors être utile d'utiliser une méthode de levée d'ambiguïté.

La figure 4.6 illustre un cas d'ambiguïté : l'entité [1] A peut être associée à l'échelle 1 et à l'échelle 2 car sa valeur mesurée appartient aux deux échelles. À titre d'exemple nous proposons les deux méthodes de levée d'ambiguïté suivantes.

1. Pour simplifier l'écriture, nous considérons A et B comme des projections d'entités isolées, le raisonnement est exactement le même pour des ensembles de projections.

La première méthode consiste à prendre en compte les centres des échelles et à associer l'entité à l'échelle dont le centre est le plus proche de sa valeur mesurée. Selon cette méthode, dans le cas illustré par la figure 4.6, l'entité A est associée uniquement à l'échelle 1. Les entités A et B sont donc associées à des échelles différentes. Cette méthode permet d'éviter d'associer une entité à une échelle lorsque la valeur mesurée de l'entité est très proche d'une borne de l'échelle.

La deuxième méthode consiste à profiter de l'ambiguïté pour réduire la complexité du système en terme de nombre d'échelles. Selon cette méthode, l'entité A peut donc être associée à l'échelle 2, ce qui permet de réduire le nombre d'échelles présentes dans le système.

Ces méthodes peuvent également être combinées. Par exemple, il est possible de décider d'appliquer de préférence la première méthode, et si la valeur mesurée est à égale distance des deux centres, ou si la différence n'est pas jugée significative, d'appliquer alors la deuxième méthode.

Le choix de l'application ou non d'une méthode de levée d'ambiguïté appartient au concepteur du système. Il peut par exemple décider de considérer qu'une entité (ou un ensemble d'entités) peut être associée à plusieurs échelles à la fois, ou choisir arbitrairement la première échelle trouvée, ou encore définir une méthode plus élaborée, comme celles que nous avons présentées.

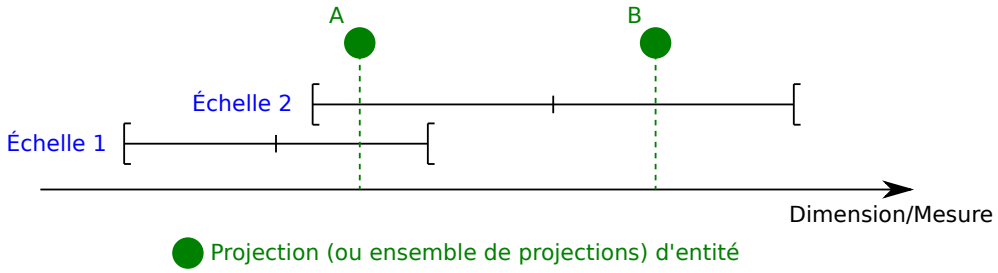

FIGURE 4.6 – Illustration d'un cas d'ambiguïté

4.2.3 Relation d'ordre entre échelles d'un ensemble d'échelles

Dans cette section, nous proposons d'ajouter une relation d'ordre entre les échelles d'un ensemble d'échelles. En effet, le terme d'échelle est intimement lié à la notion

d'ordre ou de comparaison entre échelles : les termes de petite échelle ou de grande échelle sont naturellement employés. De plus, cela étend l'expressivité du vocabulaire multiéchelle en permettant d'utiliser des opérateurs de comparaison. Par exemple, dans le cadre de la spécification d'un plan de déploiement d'un système réparti, cela permet de spécifier une contrainte comme « déployer le composant C sur les échelles *supérieures à* l'échelle giga octets ».

Ainsi, nous proposons d'ajouter des contraintes lors du choix d'un ensemble d'échelles caractéristique pour un couple (d, m) dans un point de vue v afin de pouvoir construire une relation d'ordre (partielle ou totale) entre les échelles de cet ensemble d'échelles.

Comme nous l'avons vu dans la section précédente, une échelle S est définie comme un intervalle semi-ouvert à droite de $\mathcal{M}_\mathcal{S}$ et peut s'écrire $S = [a; b[|(a, b) \in \mathcal{M}_\mathcal{S}^2$. La relation d'ordre de $\mathcal{M}_\mathcal{S}$ est notée $\leq$. À partir de ces notations nous proposons la relation binaire $\trianglelefteq$ définie de la manière suivante :

Définition 4.2. *La relation signifiant « l'échelle S est plus petite ou égale à l'échelle S' » est définie par la relation binaire $\trianglelefteq$ dans $\mathcal{I}$ telle que :*

$$\forall S = [a; b[|(a, b) \in \mathcal{M}_\mathcal{S}^2, \quad \forall S' = [a'; b'[|(a', b') \in \mathcal{M}_\mathcal{S}^2, \quad S \trianglelefteq S' \iff (a \leq a' \wedge b \leq b')$$

Cette relation binaire est une relation d'ordre car elle est réflexive, transitive, et antisymétrique. Nous définissons de manière réciproque la relation d'ordre $\trianglerighteq$.

Nous allons montrer que la relation $\trianglelefteq$ est une relation d'ordre partielle, c'est-à-dire qu'il existe des échelles non comparables entre elles. L'ensemble des couples d'échelles non comparables entre elles peut être défini de la manière suivante :

$$\{(S, S') \in \mathcal{I}^2 \quad | \quad \overline{S \trianglelefteq S'} \wedge \overline{S' \trianglelefteq S}\}$$

En notant $S = [a; b[|(a, b) \in \mathcal{M}_\mathcal{S}^2$ et $S' = [a'; b'[|(a', b') \in \mathcal{M}_\mathcal{S}^2$ nous avons les équiva-

lences suivantes :

$$\begin{aligned}
\overline{S \trianglelefteq S'} \wedge \overline{S' \trianglelefteq S} &\iff \overline{(a \leq a' \wedge b \leq b')} \wedge \overline{(a' \leq a \wedge b' \leq b)} \\
&\iff (a > a' \vee b > b') \wedge (a' > a \vee b' > b) \\
&\iff (a > a' \wedge a' > a) \vee (a > a' \wedge b' > b) \vee (b > b' \wedge a' > a) \vee (b > b' \wedge b' > b) \\
&\iff (a > a' \wedge b' > b) \vee (b > b' \wedge a' > a) \\
&\iff S \subsetneq S' \vee S' \subsetneq S
\end{aligned}$$

Ainsi nous avons démontré que l'ensemble des couples d'échelles non comparables entre elles est l'ensemble des couples (S, S') pour lesquels nous avons soit S qui est un sous-ensemble strict de S', soit S' qui est un sous-ensemble strict de S, ce qui peut s'écrire :

$$\{(S, S') \in \mathcal{I}^2 \quad | \quad S \subsetneq S' \vee S' \subsetneq S\}$$

Nous proposons donc la contrainte suivante :

Contrainte 4.4. *(C4.4) Un ensemble d'échelles $(S_i)_{i \in I}$ doit être composé uniquement d'échelles qui ne sont jamais incluses strictement l'une dans l'autre deux à deux :*

$$\forall (i, j) \in I^2, \quad \overline{S_i \subsetneq S_j} \wedge \overline{S_j \subsetneq S_i}$$

La figure 4.7 illustre la contrainte ci-dessus. Dans la partie supérieure de la figure, les échelles S et S' respectent la contrainte 4.4 et sont donc comparables pour la relation d'ordre $\trianglelefteq$: on a $S \trianglelefteq S'$. À l'inverse, dans la partie inférieure, les échelles S et S' ne respectent pas la contrainte, car $S' \subset S$, et ne sont donc pas comparables pour cette relation d'ordre.

La contrainte 4.4 implique la propriété suivante :

Propriété 4.3. *Tout ensemble d'échelles $(S_i)_{i \in I}$ respectant la contrainte 4.4 est totalement ordonné par la relation binaire $\trianglelefteq$.*

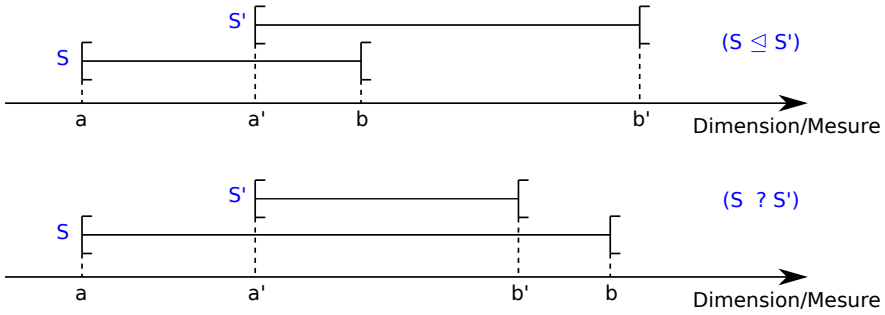

FIGURE 4.7 – Relation d'ordre entre échelles

4.2.3.1 Cas particulier pour les dimensions hiérarchiques

Dans cette section nous nous intéressons plus particulièrement aux dimensions hiérarchiques. Une échelle S est toujours définie comme un intervalle semi-ouvert à droite de $\mathcal{M}_\mathcal{S}$, c'est-à-dire comme un intervalle semi-ouvert à droite de valeurs de la mesure m. Dans le cas des dimensions hiéarchiques, et par souci de simplification, nous supposons que m est une mesure numérique ou sémantique qui mesure le *niveau hiérarchique* de la projection d'une entité. $\mathcal{M}_\mathcal{S}$ étant borné et totalement ordonné (cf. section 4.2.1), il possède une borne minimale que nous notons $MIN = 0$. Afin de prendre en compte la nature hiérarchique de la dimension étudiée, nous proposons de définir une échelle S comme *un intervalle semi-ouvert à droite de* $\mathcal{M}_\mathcal{S}$ *dont la borne inférieure est* 0. Ainsi une échelle contient tous les niveaux hiérarchiques strictement inférieurs à celui de sa borne supérieure, notée a. On note $\mathcal{I}$ l'ensemble des intervalles semi-ouverts à droite de $\mathcal{M}_\mathcal{S}$. Une échelle $S \in \mathcal{I}$ peut donc s'écrire $S = [0; a[, a \in \mathcal{M}_\mathcal{S}$.

Nous remarquons que la relation d'ordre que nous avons définie précédemment est bien applicable à des échelles de cette nature à condition que les bornes supérieures soient toutes différentes les unes des autres. En effet, un ensemble d'échelles constitué de telles échelles vérifie la contrainte 4.4.

Cependant, cette relation d'ordre n'est pas la plus intuitive dans le cas des dimensions hiérarchiques. En effet, nous pouvons remarquer que, par construction, l'ensemble des échelles S de la forme ci-dessus est naturellement ordonné par la relation binaire ensembliste d'inclusion, c'est-à-dire que :

$$\forall S = [0; a[|a \in \mathcal{M}_\mathcal{S}, \quad \forall S' = [0; a'[|a' \in \mathcal{M}_\mathcal{S}, \quad (S \subset S') \vee (S' \subset S)$$

Nous proposons donc la contrainte suivante :

Contrainte 4.5. *(C4.5) Pour toute dimension hiérarchique d associée à une mesure m à valeurs dans $\mathcal{M}_{\mathcal{S}}$, un ensemble d'échelles $(S_i)_{i\in I}$ doit être composé uniquement d'échelles de la forme $S_i = [0; a_i[$, avec 0 la borne inférieure de $\mathcal{M}_{\mathcal{S}}$:*

$$\forall i \in I, \quad S_i = [0; a_i[| a_i \in \mathcal{M}_{\mathcal{S}}$$

Cette contrainte permet d'avoir la propriété suivante :

Propriété 4.4. *Pour toute dimension hiérarchique d associée à une mesure m à valeurs dans $\mathcal{M}_{\mathcal{S}}$, un ensemble d'échelles $(S_i)_{i\in I}$ respectant la contrainte 4.5 est totalement ordonné par la relation binaire d'inclusion $\subset$.*

4.3 Conclusion

Dans ce chapitre nous avons tout d'abord présenté les concepts du vocabulaire multiéchelle que nous proposons. Nous avons défini un processus de caractérisation multiéchelle pour les systèmes répartis complexes. Nous avons proposé une formalisation des concepts d'échelle et d'ensemble d'échelles à l'aide de la théorie des ensembles. Cette formalisation nous permet notamment de proposer des contraintes que doit respecter un ensemble d'échelles afin de posséder des propriétés qui facilitent la caractérisation multiéchelle d'un système. En effet, ces propriétés permettent, par exemple, d'assurer que, pour chaque dimension de chaque point de vue de la caractérisation, toute entité du système pourra être associée à au moins une échelle, voire à une unique échelle dans le cas où l'ensemble d'échelles forme une partition de la mesure considérée. Par ailleurs, la propriété d'ordre entre les échelles d'un ensemble d'échelles permet de pouvoir comparer les échelles entre elles, ce qui augmente l'expressivité du vocabulaire multiéchelle en permettant, par exemple, d'exprimer des contraintes qui concernent l'ensemble des entités associées à toutes les échelles *supérieures à* ou *inférieures à* telle échelle.

Les résultats de ce chapitre constituent les fondements du *framework* MuSCa que nous présentons dans le chapitre suivant.

Chapitre 5

Framework MuSCa

Sommaire

Dans ce chapitre, nous présentons le *framework* MuSCa qui est le cœur de la contribution de cette thèse. Ce *framework* a pour but, d'une part, de mettre en œuvre le processus de caractérisation multiéchelle, et d'autre part de produire des artefacts logiciels adaptés à une caractérisation. Il repose sur les concepts et le processus de caractérisation multiéchelle présentés dans la chapitre précédent. Nous commençons par présenter l'approche générale du *framework* MuSCa dans la section 5.1. Cette approche s'appuie sur les méthodes de l'Ingénierie Dirigée par les Modèles (IDM) (cf. chapitre 3). Puis nous détaillons les deux activités proposées par le *framework*. Premièrement, dans la section 5.2, nous présentons les différents éléments qui permettent de représenter la caractérisation multiéchelle d'un système réparti complexe sous forme d'un modèle

exploitable par l'IDM. Deuxièmement, dans la section 5.3, nous présentons les mécanismes de génération d'artefacts permettant d'apporter, à l'exécution, la conscience des échelles des entités d'un système réparti complexe. Puis, dans la section 5.4, nous détaillons l'implémentation du *framework*. Enfin, nous concluons ce chapitre dans la section 5.5.

5.1 Approche générale basée sur l'IDM

L'approche générale que nous proposons à travers le *framework* MuSCa est illustrée figure 5.1. Ce *framework* propose, d'une part, une méthodologie pour la phase de conception des systèmes répartis complexes basée sur l'analyse multiéchelle de l'architecture de ces systèmes, et d'autre part, un mécanisme de production d'artefacts logiciels qui apportent la conscience des échelles aux composants de ces systèmes au cours de la phase d'exécution. Nous avons choisi de proposer un *framework* qui s'appuie sur l'IDM, car cette démarche, et l'ensemble des outils existants qui l'implémentent (générateurs de code, langages de transformation de modèles, etc.) permettent d'établir un lien direct entre l'application du processus de caractérisation multiéchelle, au cours de la conception d'un système, et la production d'artefacts exécutables.

La phase de conception, détaillée sous forme d'un diagramme SPEM [OMG 2008b] dans la partie gauche de la figure 5.1, est composée de deux activités principales.

La première activité est la caractérisation multiéchelle, qui consiste à abstraire une vision multiéchelle d'un futur système distribué complexe. Afin de guider le concepteur du système et de capitaliser sur les anciennes caractérisations, elle prend en entrée un document partagé que nous appelons la taxonomie multiéchelle. Cette taxonomie contient l'ensemble des éléments de caractérisation ayant déjà été utilisés pour d'anciennes caractérisations. De plus elle est extensible, ce qui permet au concepteur du système d'ajouter les éventuels nouveaux termes de caractérisation dont il peut avoir besoin, contribuant ainsi à la capitalisation d'une connaissance partagée. Ces termes seront alors accessibles pour de futures caractérisations. Cette première activité produit en sortie une caractérisation multiéchelle du système étudié, sous forme d'un modèle, qui correspond à une restriction de la taxonomie multiéchelle.

La deuxième activité est la génération d'artefacts logiciels qui apportent la conscience

multiéchelle aux composants du système à l'exécution. Nous appelons ces artefacts des sondes multiéchelles. Cette activité prend en entrée le modèle de caractérisation multiéchelle produit par la première activité, ainsi que des sondes logicielles de bas niveau. Ces dernières sont des programmes qui s'exécutent sur les entités du système afin de mesurer les caractéristiques de ces entités dans différentes dimensions. Par exemple, pour le point de vue équipement, une sonde de bas niveau peut retourner la capacité de stockage ou la puissance de calcul d'un équipement. Nous considérons que, pour chaque dimension choisie dans la caractérisation d'un système, il existe une sonde de bas niveau qui mesure cette dimension. Les sondes multiéchelles générées au cours de cette deuxième activité consolident les données fournies par les sondes de bas niveau afin d'apporter la conscience des échelles aux composants du système à l'exécution.

Ces deux activités sont détaillées dans les deux sections suivantes (5.2 et 5.3).

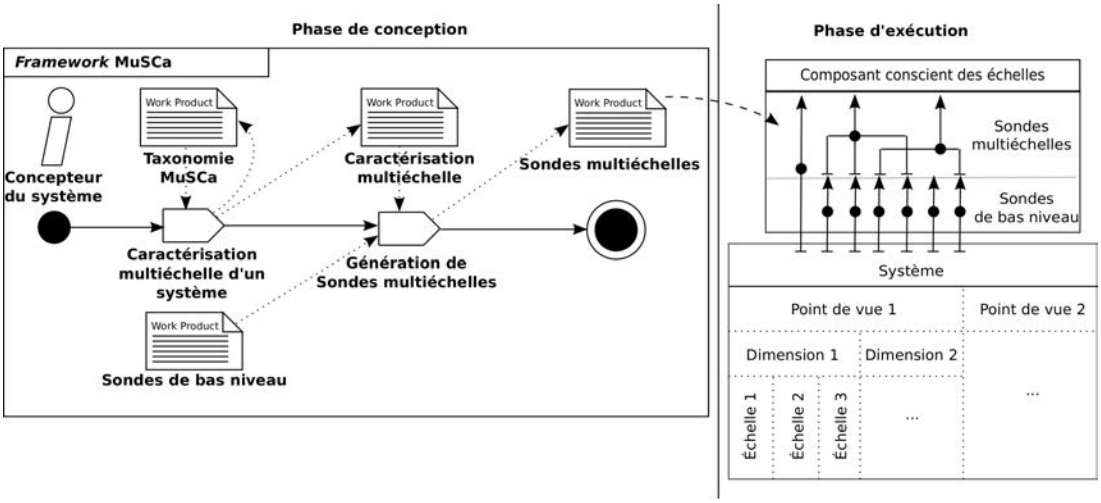

FIGURE 5.1 – Approche générale du *framework* MuSCa

5.2 Du métamodèle aux modèles MuSCa

Afin de pouvoir appliquer le processus de caractérisation multiéchelle à la conception et au déploiement de systèmes distribués conscients de l'échelle, nous avons choisi de suivre une approche dirigée par les modèles (basée sur les quatre couches de métamodélisation de l'OMG [OMG 2002], cf. section 3.2.1.1). La figure 5.2 présente les

correspondances entre les niveaux de modélisation de l'IDM et les niveaux du *framework* MuSCa. Le métamodèle MuSCa (niveau M2) est conforme au métamétamodèle Ecore [1] (niveau M3). Les classes du métamodèle MuSCa représentent les concepts du vocabulaire multiéchelle. Ce métamodèle est utilisé pour définir des modèles MuSCa (niveau M1), qui correspondent à des caractérisations multiéchelles. Ces caractérisations peuvent abstraire un ou plusieurs systèmes répartis du monde réel (niveau M0).

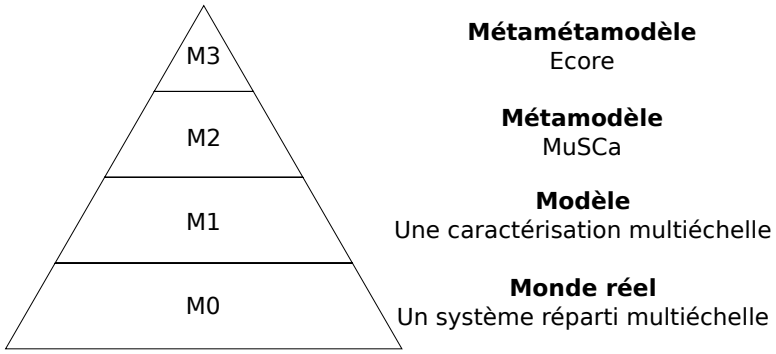

FIGURE 5.2 – MuSCa : niveaux de modélisation de l'IDM

Les trois sections suivantes présentent, respectivement, le métamodèle MuSCa, la représentation d'une caractérisation multiéchelle sous forme d'un modèle MuSCa, et la taxonomie MuSCa qui fournit un catalogue consolidé d'éléments utilisés pour les caractérisations multiéchelles.

5.2.1 Métamodèle MuSCa

Le métamodèle MuSCa est présenté figure 5.3. Ce métamodèle est basé sur le vocabulaire utilisé dans le processus de caractérisation multiéchelle (cf. sections 4.1.1 et 4.1.4).

Une instance de la classe `MSCharacterization` est obtenue à l'issue d'un processus de caractérisation. Une caractérisation considère plusieurs instances de la classe `Viewpoint`, par exemple les points de vue `Geography`, `User`, `Device` ou `Network` (classes de niveau M1). Chaque point de vue détermine une vue restreinte du système, qui est étudiée indépendamment des autres vues. Une vue d'un système (depuis un point de vue donné) est étudiée à travers différentes dimensions, représentées par des instances

1. http://www.eclipse.org/modeling/emf/

de la classe de niveau M2 `Dimension`, qui sont des caractéristiques mesurables des entités qui composent la vue. Par exemple, le point de vue équipement (classe `Device` de niveau M1) peut être analysé à travers la dimension de la capacité de stockage des équipements du système (classe `StorageCapacity` de niveau M1). Comme mentionné précédemment, une dimension est mesurable, ce qui signifie qu'elle peut être associée à une ou plusieurs mesures, représentées par des instances de `Measure`. Par exemple, au niveau M1, la dimension `StorageCapacity` peut être mesurée à l'aide de la mesure `Bytes` ou de la mesure `KiloBytes`. Pour l'association d'une dimension et d'une mesure, le concepteur du système définit une instance de `ScaleSet`, qui représente un ensemble ordonné d'échelles pertinentes pour cette caractérisation du système. Une échelle est définie par ses bornes *min* et *max*.

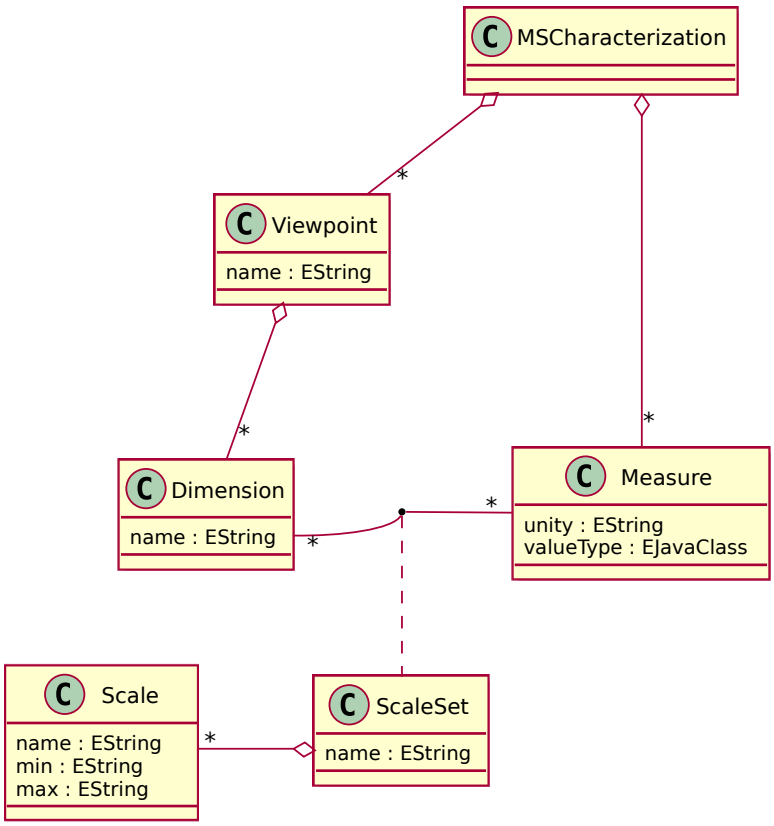

FIGURE 5.3 – MuSCa : métamodèle simplifié de caractérisation multiéchelle

Enfin, pour certains points de vue ou certaines dimensions, un système peut présenter plusieurs instances d'une même échelle. Par exemple, si l'on considère le point

de vue `Geography`, avec la dimension `AdministrativeLocation`, l'échelle `City` (niveau M1) possède plusieurs instances (niveau M0) qui représente les différentes villes dans lesquelles sont présentes des entités du système.

5.2.2 Exemple de modèle MuSCa ou caractérisation multiéchelle

Une caractérisation multiéchelle d'un système donné est le résultat de l'application du processus présenté section 4.1.4. Cette caractérisation peut être représentée sous la forme d'un modèle MuSCa conforme au métamodèle MuSCa. La figure 5.4 illustre un extrait de modèle MuSCa qui est le résultat de l'application du processus de caractérisation multiéchelle au scénario présenté section 1.4. Quatre points de vue ont été selectionnés : les points de vue équipement, réseau, géographie et utilisateur (la figure ne montre que les dimensions et échelles relatives au point de vue géographie).

Nous étudions le point de vue géographie à travers deux dimensions. La première dimension est la localisation. Elle mesure la répartition des entités du système en étudiant les localisations administratives communes à un ensemble d'entités conscientes des échelles. Pour cette dimension, mesurée à l'aide de la mesure que nous appelons « niveau de localisation administrative » (mesure sémantique), nous identifions les échelles suivantes : immeuble, quartier, ville, région, pays et monde. La seconde dimension est la distance. Elle mesure la répartition des entités du système en étudiant les distances qui les séparent les unes des autres. Pour cette dimension, mesurée en mètres (mesure numérique), nous identifions les échelles suivantes : locale, accessible à pied, accessible en voiture et accessible en avion. Ces deux dimensions sont des dimensions hiérarchiques : une ville est incluse dans un pays, une zone accessible à pied est incluse dans une zone accessible en voiture, etc. Les échelles peuvent donc être des intervalles de la forme $[0; max[$ (cf. section 4.2.3.1). Une illustration de ces deux dimensions a été proposée dans la section 4.1.2.4, figure 4.4.

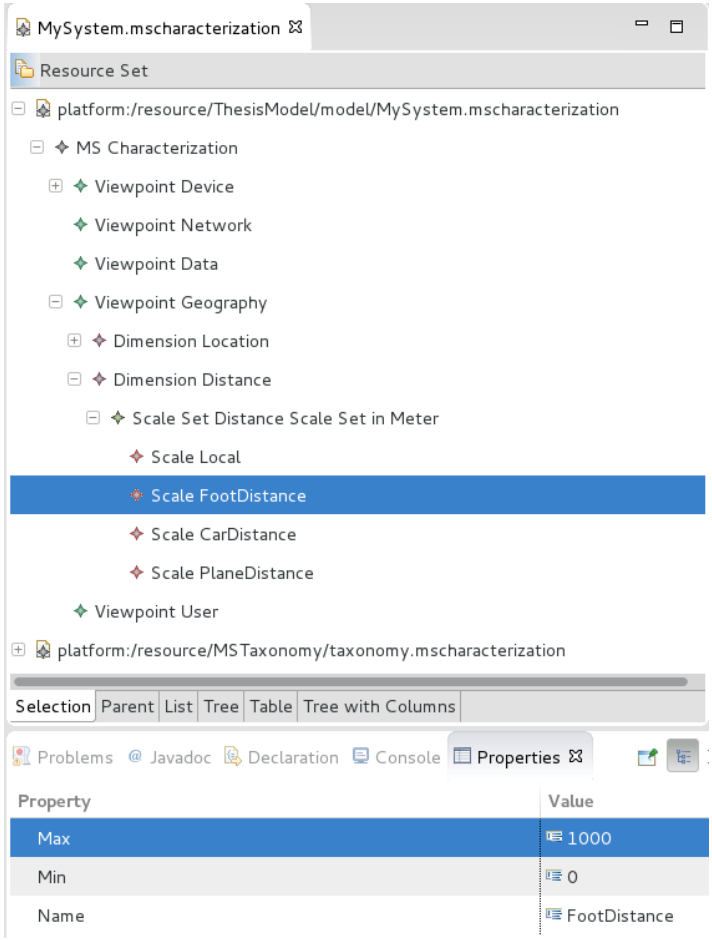

FIGURE 5.4 – Édition d'un modèle MuSCa

5.2.3 Taxonomie MuSCa : catalogue d'éléments de caractérisation multiéchelle

La taxonomie MuSCa est un modèle de niveau M1 qui regroupe l'ensemble des éléments multiéchelles collectés lors de précédentes caractérisations. C'est un document partagé et extensible qui permet de capitaliser sur les caractérisations multiéchelles déjà réalisées. Une nouvelle caractérisation peut être construite par restriction de la taxonomie MuSCa à laquelle il est possible d'ajouter de nouveaux éléments pertinents pour le système en cours d'étude. Ces éléments pourront alors être ajoutés à la taxonomie afin de pouvoir être directement réutilisés lors d'une prochaine caractérisation.

La taxonomie MuSCa est conforme au métamodèle MuSCa. Un exemple de taxonomie en format XML est consultable en annexe A.

5.3 Génération d'artefacts pour la conscience des échelles

Dans la section précédente, nous avons montré que le *framework* MuSCa permet tout d'abord de formaliser les différents concepts du vocabulaire multiéchelle sous la forme d'un métamodèle, puis de définir une caractérisation multiéchelle d'un système réparti complexe donné, et enfin de constituer une base de connaissance sous forme d'une taxonomie d'éléments de caractérisation multiéchelle. Ainsi, le *framework* MuSCa permet aux concepteurs d'un système réparti complexe de définir un modèle représentant la caractérisation multiéchelle du système qu'ils souhaitent concevoir, en s'appuyant sur une base de connaissance constituée lors de précédentes caractérisations, et/ou en la complétant.

La prochaine étape de l'approche IDM est d'exploiter un modèle de caractérisation pour produire des artefacts utilisables au cours de l'exécution du système modélisé. En particulier, nous proposons de générer des artefacts de conscience des échelles que nous appelons des sondes multiéchelles. Dans les sections suivantes, nous commençons par définir la notion de conscience des échelles, puis nous présentons les mécanismes de génération des sondes multiéchelles.

5.3.1 Définition de la conscience des échelles

Jusqu'à présent, nous avons présenté la vision multiéchelle des systèmes répartis complexes comme un outil d'analyse et de conception de ces systèmes. Cependant, nous pensons que cette vision peut également être utile au cours de l'exécution d'un système. C'est pourquoi nous introduisons la notion de conscience des échelles.

Définition 5.1. *Une entité —ou un ensemble d'entités— d'un système est consciente des échelles si elle a la capacité de savoir, à l'exécution, à quelles échelles elle est associée dans les différents points de vue et dimensions retenus lors de la caractérisation du système auquel elle appartient.*

Cette définition implique qu'une entité consciente des échelles doit posséder un moyen de découvrir les échelles auxquelles elle est associée. Nous proposons donc d'équiper ces entités d'artefacts logiciels que nous appelons des sondes multiéchelles. Ces sondes peuvent être appelées par l'entité à laquelle elles sont liées et renvoient des informations concernant les échelles, et le cas échéant les instances d'échelles (niveau M0), auxquelles l'entité est associée.

Les domaines d'application de la conscience des échelles sont variés. Dans le chapitre 6, nous détaillons deux cas d'utilisation de notre approche : le déploiement de systèmes répartis multiéchelles, et le filtrage d'information de contextes dans des systèmes répartis multiéchelles.

5.3.2 D'un modèle MuSCa aux sondes multiéchelles

Un modèle MuSCa contient l'ensemble des éléments de caractérisation multiéchelle pertinents pour le système en cours de conception. Il est donc propre à un système donné. C'est pourquoi, afin de rendre les entités d'un système conscientes des échelles, il est nécessaire de produire des sondes multiéchelles adaptées à la caractérisation multiéchelle de ce système, c'est-à-dire à son modèle MuSCa.

Remarque. *Comme nous l'avons présenté dans la section 3.2.2, l'IDM propose deux méthodes pour rendre un modèle productif : la transformation de modèles (transformation M2M), et la génération de code (transformation M2T). Ces méthodes peuvent être combinées, ce qui implique plusieurs approches possibles. Une première approche envisageable consiste à générer directement le code des sondes multiéchelles à partir d'un modèle MuSCa en utilisant un générateur de code. C'est l'approche que nous présentons dans cette section car elle a été complètement implémentée.*

L'objectif est donc de pouvoir générer les sondes multiéchelles de façon automatique, c'est-à-dire sans qu'il soit nécessaire qu'un programmeur complète le code généré. Pour cela nous proposons un deuxième métamodèle, appelé MuSCa_Probes, qui est une extension de MuSCa avec des concepts techniques nécessaires à l'implémentation des sondes. Une version simplifiée de ce métamodèle est présentée figure 5.5. Afin de ne pas surcharger la figure, celle-ci ne présente pas les classes `ScaleSet` et `Scale` qui ne sont pas étendues dans MuSCa_Probes. Ce métamodèle possède principalement les extensions suivantes :

— Une classe `BasicProbe` qui représente les sondes de bas niveau (cf. section 5.1). Chaque `Dimension` référence la `BasicProbe` qui permet de mesurer une entité pour cette dimension. Chaque `BasicProbe` référence une `Measure` qui indique l'unité de la valeur retournée par la sonde de bas niveau. Cette classe possède, en attributs, l'ensemble des informations nécessaires à l'utilisation de la sonde (classe principale, paramètres à passer au constructeur, initialisation éventuelle, etc.) ;

— Une classe `MavenArtifact` qui représente des artefacts Maven [2]. Chaque `BasicProbe` référence un `MavenArtifact`, ce qui permet de gérer automatiquement les dépendances entre les sondes multiéchelles et les sondes de bas niveau. Ceci nécessite d'implémenter les sondes de bas niveau sous forme de modules Maven ;

— Une classe `MeasureConverter` qui permet de spécifier des formules de conversion d'une mesure à l'autre. Cette classe est utile pour transformer une valeur mesurée par une sonde basique en une valeur de mesure d'un ensemble d'échelles particulier. En effet, la mesure d'un ensemble d'échelles —qui définit l'unité des bornes *min* et *max* de ses échelles— peut être différente de celle associée à la sonde de bas niveau de la dimension de cet ensemble d'échelles, ce qui nécessite un mécanisme de conversion ;

— La classe `Viewpoint` possède un attribut supplémentaire qui contient la liste des propriétés d'une projection d'entité dans ce point de vue : par exemple les coordonnées GPS pour le point de vue géographie.

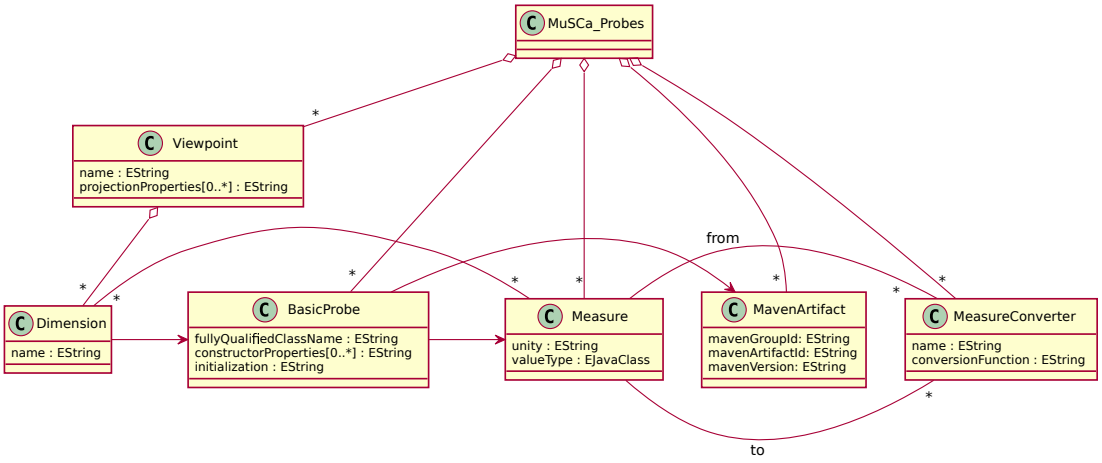

FIGURE 5.5 – Métamodèle MuSCa_Probes simplifié

2. http://maven.apache.org/

Grâce à ces extensions, nous sommes en mesure de spécifier un générateur capable de produire complètement les sondes multiéchelles correspondant à un modèle MuSCa_Probes (cf. figure 5.6). Une sonde est générée pour chaque point de vue présent dans la caractérisation du système étudié. Chaque sonde expose une ou plusieurs méthodes par dimension. En fonction du type de chaque dimension (cf. section 4.1.3), les méthodes exposées sont différentes.

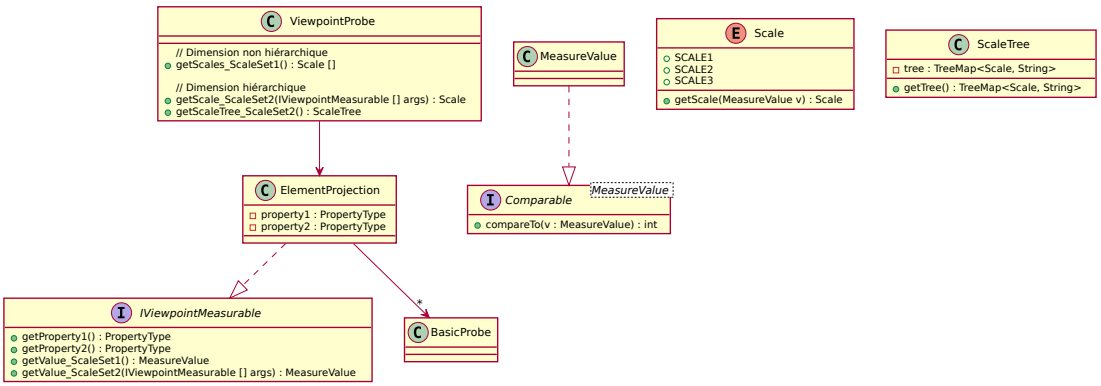

FIGURE 5.6 – Diagramme de classes des sondes multiéchelles générées par le *framework* MuSCa pour une caractérisation

Nous distinguons les dimensions hiérarchiques et non hiérarchiques (cf. section 4.1.3) pour définir les méthodes exposées. Premièrement, pour chaque dimension non hiérarchique, une sonde expose une unique méthode qui retourne la liste des échelles associées à l'entité sur laquelle la sonde est déployée. En effet, comme nous l'avons vu dans la section 4.2, un ensemble d'échelles doit former un recouvrement de l'ensemble des valeurs de sa mesure mais pas nécessairement une partition. Il est donc possible qu'une entité soit associée à plusieurs échelles. Le nombre d'échelles retournées dépend de la méthode de levée d'ambiguïté éventuellement choisie (cf. section 4.2.2).

Deuxièmement, pour chaque dimension hiérarchique, une sonde expose plusieurs méthodes :

— Une première méthode qui retourne un tableau d'associations échelle/instance d'échelle, que nous appelons un `ScaleTree`, qui représente les instances d'échelles auxquelles est associée l'entité. Par exemple, les figures 5.7 et 5.8 présentent les `ScaleTree` retournés pour des entités situées respectivement dans le centre ville de Toulouse et à l'Université Paul Sabatier ;
— Une deuxième méthode qui retourne la plus petite échelle commune à l'entité

locale, c'est-à-dire celle sur laquelle la sonde est déployée, et un ensemble d'entités distantes. Par exemple, la plus petite échelle commune aux entités dont les `ScaleTree` sont présentés dans les figures 5.7 et 5.8 est l'échelle ville (*CITY*) dont l'instance commune est *Toulouse*.

```
1 {POINT=43.602905;1.442682, NEIGHBORHOOD=Capitole, SUBURB=Toulouse Centre,
    CITY=Toulouse, REGION=Midi-Pyrénées, COUNTRY=fr, WORLD=World}
```

FIGURE 5.7 – `ScaleTree` d'une entité située au centre de Toulouse (dimension localisation administrative, point de vue géographie)

```
1 {POINT=43.562894;1.46616, STREET=Voie 20, NEIGHBORHOOD=Rangueil, Sauzelong,
    Pech David, Pouvourville, SUBURB=Toulouse Sud-Est, CITY=Toulouse, REGION=
    Midi-Pyrénées, COUNTRY=fr, WORLD=World}
```

FIGURE 5.8 – `ScaleTree` d'une entité située à l'Université Paul Sabatier de Toulouse (dimension localisation administrative, point de vue géographie)

5.4 Implémentation du *framework* MuSCa

Dans cette section nous présentons les travaux d'implémentation du *framework* MuSCa. La première section présente l'implémentation de la phase de modélisation permettant d'appliquer le processus de caractérisation multiéchelle. La seconde section présente l'implémentation du mécanisme de génération des sondes multiéchelles pour la conscience des échelles.

5.4.1 Éditeur de caractérisation multiéchelle

Nous avons implémenté le *framework* MuSCa à l'aide du *Eclipse Modeling Framework Project*[3] (EMF). Les métamodèles MuSCa et MuSCa_Probes sont définis comme des instances du métamétamodèle Ecore. EMF génère automatiquement un éditeur de modèles MuSCa. Nous avons étendu cet éditeur pour y ajouter un assistant qui permet de créer un modèle de caractérisation en sélectionnant des points de vue, dimensions, et ensembles d'échelles présents dans la taxonomie MuSCa (cf. figures 5.9, 5.10, et 5.11). Ce modèle peut ensuite être complété via l'éditeur généré par EMF.

3. http://www.eclipse.org/modeling/emf/

L'éditeur de caractérisation multiéchelle est empaqueté sous la forme d'un *plugin*[4] de l'environnement de développement Eclipse[5].

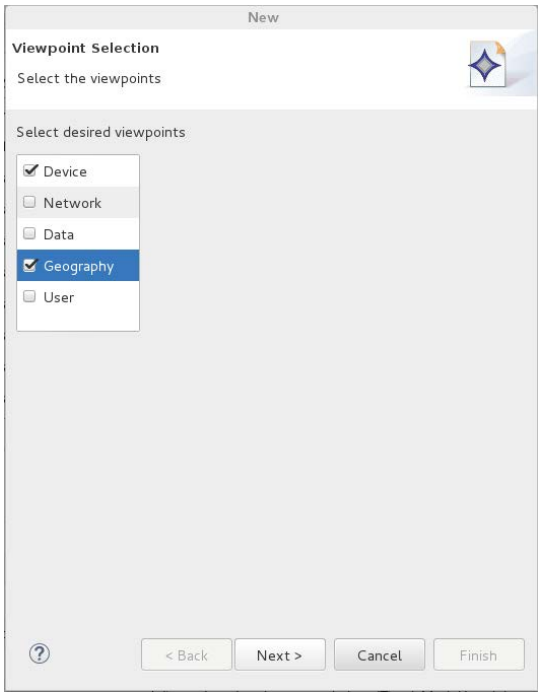

FIGURE 5.9 – Assistant de sélection des points de vue

4. Nous utilisons le terme anglais « plugin » qui est plus couramment utilisé que l'expression française « module d'extension ».
5. https://eclipse.org/

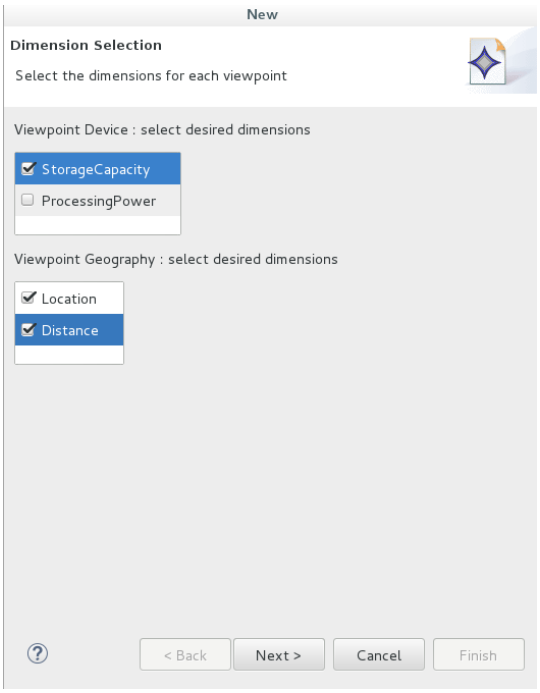

FIGURE 5.10 – Assistant de sélection des dimensions

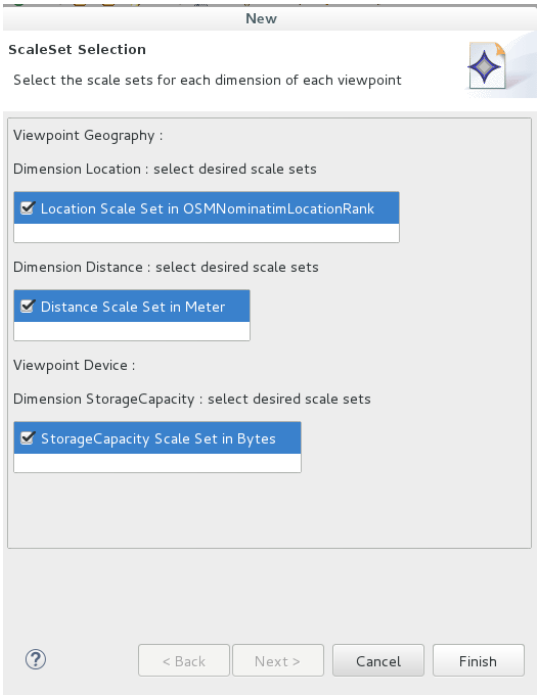

FIGURE 5.11 – Assistant de sélection des ensembles d'échelles

5.4.2 Conscience des échelles

Pour générer les sondes multiéchelles nous utilisons le générateur de code Acceleo[6] qui implémente le standard MOF Model To Text Transformation Language (MOFM2T) de l'OMG [OMG 2008a]. Pour cela nous avons défini des *templates* Acceleo qui contiennent une implémentation générique des sondes multiéchelles (cf. annexe B). Ces *templates* sont complétés par le générateur de code pour produire le code des sondes multiéchelles correspondant à un modèle donné en entrée. Nous avons choisi de générer des sondes implémentées en Java. Les *templates* ainsi que les classes nécessaires au lancement de la génération de code sont empaquetés sous la forme d'un deuxième *plugin* pour l'environnement Eclipse.

Après avoir installé ce *plugin*, la génération de code peut être lancée en accédant au menu contextuel (clique droit) lié à un modèle de caractérisation multiéchelle (cf. figure 5.12).

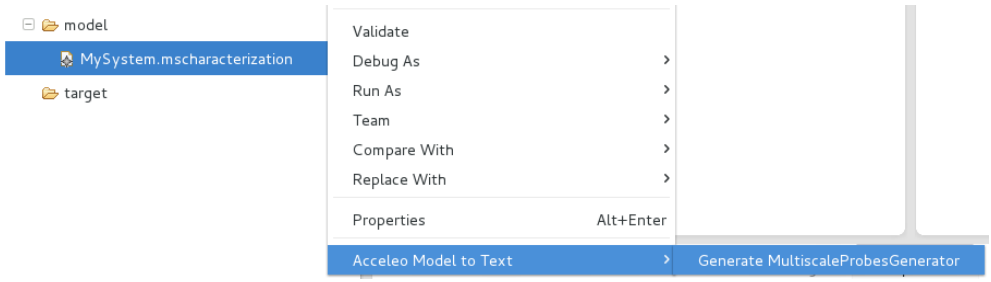

FIGURE 5.12 – Menu contextuel Acceleo

Un extrait d'une sonde automatiquement générée est présenté dans la figure 5.13. Cette figure présente l'énumération générée pour l'ensemble d'échelles associé à la dimension distance et à la mesure mètres. Cette énumération contient notamment, lignes 2 à 5, les différentes échelles qui peuvent être retournées par la sonde multiéchelle pour l'ensemble d'échelles correspondant. Elle contient également deux méthodes : une méthode statique `getScale`, ligne 15, qui retourne la première échelle contenant la valeur mesurée passée en paramètre, et une méthode `isInScale`, ligne 24, qui détermine si une valeur mesurée est incluse dans une échelle.

Remarque. *La méthode* `getScale` *retourne toujours la première échelle trouvée. Cependant, il est possible de la modifier après génération afin d'implémenter une méthode*

6. http://www.eclipse.org/acceleo/

de levée d'ambiguïté lorsque plusieurs échelles contiennent une même valeur mesurée (cf. section 4.2.2).

La figure 5.14 présente un extrait de *template* Acceleo. Les versions complètes de ce *template* et des principaux autres *templates* que nous avons définis sont consultables en annexe B. Lorsque ce *template* est appliqué au modèle MuSCa présenté dans la figure 5.4, section 5.2.2, cela génère les échelles listées entre les lignes 2 et 5 de la figure 5.13.

```
1  public enum Distance_In_Meter_Scale {
2      LOCAL("0", "10"),
3      FOOTDISTANCE("0", "1000"),
4      CARDISTANCE("0", "100000"),
5      PLANEDISTANCE("0", "Infinity");

7      private final Meter_Value min;
8      private final Meter_Value max;

10     Distance_In_Meter_Scale(String min, String max) {
11         this.min = new Meter_Value(min);
12         this.max = new Meter_Value(max);
13     }

15     public static Distance_In_Meter_Scale getScale(Meter_Value value) {
16         for (Distance_In_Meter_Scale scale : Distance_In_Meter_Scale.values()) {
17             if (scale.isInScale(value)) {
18                 return scale;
19             }
20         }
21         return null;
22     }

24     private boolean isInScale(Meter_Value value) {
25         return (value.compareTo(this.min) >= 0) && (value.compareTo(this.max) <= 0);
26     }
27 }
```

FIGURE 5.13 – Extrait de sonde générée : énumération Distance_In_Meter_Scale

```
1 [for (scale : Scale | aScaleSet.scales) separator(',\n')]
2     [scale.name.toUpper()/]("[scale.min/]", "[scale.max/]")
3 [/for];
```

FIGURE 5.14 – Extrait de *template* Acceleo

Évaluation du nombre de lignes générées Pour une caractérisation contenant trois ensembles d'échelles (associés aux dimensions distance et localisaton administrative pour le point de vue géographie, et à la dimension capacité de stockage pour le

point de vue équipement), le processus de génération de sondes multiéchelles produit 991 lignes de code.

5.5 Conclusion

Dans ce chapitre nous avons présenté le *framework* MuSCa qui constitue le cœur de la contribution de cette thèse. Ce *framework* repose sur une démarche d'IDM.

Premièrement, il permet au concepteur d'un système réparti complexe de mettre en œuvre le processus de caractérisation multiéchelle —proposé et formalisé dans le chapitre précédent— à l'aide d'un éditeur de caractérisation multiéchelle. Cet éditeur, qui fait partie du *framework* MuSCa, a été généré à partir d'un métamodèle qui constitue une représentation formelle et informatisée des concepts du vocabulaire multiéchelle que nous avons proposé dans le chapitre précédent. Pour guider le concepteur, l'éditeur propose un assistant qui intègre une taxonomie multiéchelle qui contient tous les éléments de caractérisation déjà identifés lors de précédentes caractérisations. Le concepteur a également la possibilité d'étendre cette taxonomie. Grâce à cet éditeur, un concepteur de système peut définir une caractérisation multiéchelle du système qu'il souhaite concevoir. Une telle caractérisation est un modèle conforme au métamodèle MuSCa.

Deuxièmement, à partir d'un modèle de caractérisation MuSCa, le *framework* permet de générer des artefacts logiciels, que nous appelons des sondes multiéchelles, qui apportent la conscience des échelles aux composants du système caractérisé. Le *framework* met donc en œuvre la démarche IDM en rendant productif un modèle de caractérisation. Les sondes multiéchelles sont générées spécifiquement pour une caractérisation donnée, et donc pour un système donné, ce qui permet de garantir la concordance des échelles pouvant être retournées par les sondes avec les échelles définies dans une caractérisation. Ces sondes multiéchelles sont déployées à travers l'ensemble du système et permettent aux entités, ou ensembles d'entités, de savoir à tout moment à quelle(s) échelle(s) elles ou ils sont associé(e)s.

Le respect d'une démarche IDM permet d'augmenter l'intérêt du processus de caractérisation multiéchelle en rendant productifs les modèles de caractérisation. De plus, l'IDM garantit une évolutivité aisée du *framework*. Par exemple, s'il est nécessaire de

modifier le métamodèle MuSCa, l'éditeur de caractérisation peut être automatiquement regénéré et mis à jour pour refléter ces changements. Par ailleurs, de nombreux outils existent pour implémenter cette démarche, ce qui réduit l'effort de développement nécessaire à l'implémentation du *framework*.

Dans le chapitre suivant, nous validons le processus de caractérisation ainsi que le *framework* MuSCa en présentant trois cas d'utilisation.

Chapitre 6

Validation du *framework* MuSCa à travers des cas d'utilisation

Sommaire

Dans ce chapitre nous validons notre approche à travers deux axes. D'une part, nous validons l'expressivité du vocabulaire multiéchelle et du métamodèle MuSCa ainsi que l'apport de la taxonomie MuSCa lors de la caractérisation multiéchelle d'un système. D'autre part, nous validons le code produit par notre approche IDM, et en particulier les sondes multiéchelles, en montrant que ces sondes permettent d'apporter la conscience des échelles à l'exécution.

Nous évaluons ces deux axes à travers trois travaux de recherche qui utilisent le *framework* MuSCa. Ces travaux sont mis en œuvre dans le cadre du projet ANR IN-

COME[1]. L'objectif principal du projet est de concevoir et mettre en œuvre un *middleware* de gestion de contexte multiéchelle pour l'*IoT*. Tout d'abord, dans la section 6.1, nous présentons le travail de caractérisation de scénarios multiéchelles qui a été effectué au début du projet INCOME. Ce travail avait pour but de cibler précisément les exigences de conception du gestionnaire de contexte, et notamment les exigences liées au multiéchelle. Puis, dans la section 6.2, nous présentons les travaux qui concernent le déploiement d'un système de gestion de contexte multiéchelle, et plus généralement d'un système réparti multiéchelle. Nous présentons ensuite, dans la section 6.3, les travaux liés à la gestion de la diffusion d'informations de contexte au sein d'un gestionnaire de contexte multiéchelle. Enfin, dans la section 6.4, nous présentons le bilan de la validation de notre approche à travers les deux axes de validation définis plus haut.

6.1 Caractérisation multiéchelle des scénarios INCOME

Dans cette section nous présentons l'utilisation du vocabulaire multiéchelle dans le cadre de la caractérisation multiéchelle des scénarios INCOME. Nous commençons, en section 6.1.1, par présenter l'utilisation du vocabulaire multiéchelle comme une grille de lecture pour la caractérisation multiéchelle de scénarios existants de l'*IoT*. Nous illustrons cette étude en détaillant un des scénarios étudiés. Puis, en section 6.1.2, nous discutons de l'extensibilité de la taxonomie et du *framework* MuSCa.

6.1.1 Caractérisation multiéchelle comme grille de lecture de scénarios existants

Le vocabulaire MuSCa a été utilisé comme grille de lecture dans le cadre du projet INCOME, dans le but de caractériser la nature multiéchelle de plusieurs scénarios existants de l'*IoT* [Arcangeli 2013b]. Ce travail a été réalisé à partir d'une première version de la taxonomie MuSCa qui contenait les principaux points de vue, dimensions, mesures et échelles que nous avions identifiés jusqu'alors. Les scénarios ont été analysés à travers ces différents points de vue et dimensions pour mettre en avant les échelles pertinentes qu'ils faisaient intervenir. Cette approche a permis aux membres du projet INCOME de comparer une grande variété de scénarios.

1. http://www.irit.fr/income/

6.1. Caractérisation multiéchelle des scénarios INCOME

Nous présentons ci-dessous la grille de lecture qui a été utilisée dans le cadre du projet INCOME pour caractériser des scénarios existants. Par souci de simplification, une seule dimension a été implicitement retenue pour chaque point de vue étudié. Ceci permet de n'avoir a considérer qu'un seul ensemble d'échelles par point de vue. Les paragraphes suivants qui présentent les ensembles d'échelles retenus pour les différents points de vue étudiés sont extraits du livrable *Scénarios et Exigences* du projet INCOME [Arcangeli 2013b] :

Équipement : nous recensons les équipements qui participent à la réalisation des scénarios. Les équipements sont classés dans les échelles suivantes :

- *objet taggé* (objet identifiable, doté d'une puce RFID simple sans traitement local),
- *objet intelligent* (capteur / actuateur doté de plus ou moins d'intelligence),
- *équipement personnel* (téléphone mobile, tablette, ordinateur portable ou fixe),
- *cloudlet* (serveur de proximité),
- *cloud* (serveur dans les nuages).

Réseau : nous recensons les réseaux qui participent à la réalisation de ces scénarios. Les réseaux sont classés dans les échelles suivantes :

- *réseau personnel*,
- *réseau local*,
- *réseau métropolitain*,
- *réseau Internet*.

Donnée : nous recensons les types de données échangées et stockées pour la réalisation du scénario. Les types de données sont classés dans les échelles suivantes qui correspondent à un mode d'organisation des données plus ou moins structuré :

- *atomique*,
- *structurée*,
- *BD relationnelle*,
- *base de connaissance*,
- *big data*.

Géographie : nous recensons les organisations géographiques qui regroupent des entités participant aux scénarios (utilisateurs, équipements, capteurs). Les organisations

géographiques sont classées dans les échelles suivantes (organisation hiérarchique) :

— *salle*,
— *bâtiment*,
— *ville*,
— *région*,
— *monde*.

Utilisateur : nous recensons les groupes d'utilisateurs du scénario. Les groupes d'utilisateurs sont classés dans les échelles suivantes :

— *individu*,
— *groupe*,
— *communauté*,
— *tous*.

Administrative : nous recensons la répartition des acteurs du scénario dans des organisations administratives. Les organisations administratives sont classées dans les échelles suivantes :

— *individu*,
— *organisation administrative*,
— *inter-organisation*.

Cette grille de lecture a été appliquée à plusieurs scénarios existants. Nous présentons ici l'un de ces scénarios afin d'y appliquer ensuite la grille de lecture présentée précédemment. La description qui suit du scénario « Future Internet 2020 » est extraite du même livrable [Arcangeli 2013b] du projet INCOME :

Document de référence : Ce scénario est décrit dans le document [Hourcade 2009]. Il est issu d'une initiative de l'Union Européenne en 2009 qui a réuni un groupe d'experts pour imaginer les évolutions de l'Internet à l'horizon de 2020. Le document et d'autres informations sont accessibles en ligne [2].

Résumé : La famille Smith part en vacances au ski. Durant leur voyage en voiture, ils sont assistés par un système embarqué dans la voiture : guidage GPS, routes alternatives

2. Site internet projet Future Internet 2020. http://www.future-internet.eu/home.html.

selon les préférences du conducteur. De même, il propose aux passagers d'obtenir des informations sur les points d'intérêts le long du parcours.

La chambre d'hôtel est personnalisée selon les préférences des utilisateurs : lumière, température, couleurs de tapisserie (iTapestry), configuration de la télévision, etc. L'accès aux préférences utilisateurs est autorisé lors de l'enregistrement à l'arrivée à l'hôtel. L'hôtel offre des services pour la préparation des parcours de ski en fonction des profils sportifs des utilisateurs et de l'état des pistes.

Durant la journée, des informations sur les descentes sont captées par les skis (positions, accélération, vitesse). Les descentes peuvent être visualisées et partagées entre les membres de la famille. Des caméras prennent des photos / films pendant la journée et ces photos sont marquées (localisation, horodatage). De retour de vacances, photos et vidéos des parcours de skis peuvent être montrées aux amis.

Éléments d'analyse : Les éléments clefs mis en avant par ce scénario sont :

— la mise en œuvre à l'échelle de l'Internet des objets par un nombre important de capteurs / actuateurs situés dans l'environnement (route, murs, skis, capteurs biodégradables dans la neige, etc.),
— l'exemple d'un environnement sans séparation (« sans couture ») des objets physiques, informations et services,
— le partage d'informations contrôlées par l'utilisateur (respect de la vie privée).

Expérimentations réalisées : Aucune puisque le scénario est uniquement anticipatif et n'a pas donné lieu à mise en œuvre.

Analyse multiéchelle du scénario : Ce scénario utilise des capteurs embarqués passifs (RFID, capteurs divers sur les skis, etc.). Les informations sont principalement traitées à distance à partir de serveurs (informations de préférences en provenance de la maison, carnets de route, informations personnelles collectées durant les vacances). Les données traitées sont personnelles, structurées et référencées (localisation, horodatage). Le scénario se situe dans un espace large : maison, déplacements routiers, serveurs divers, station de ski. Plusieurs organisations administratives sont impliquées dans ce scénario : l'utilisateur, les données personnelles (relatives à la maison et accédées à distance), serveurs de stockage d'informations personnelles, différents serveurs d'informations publiques, serveur de données de la station. Les utilisateurs définissent

explicitement les accès à leurs données personnelles. Le scénario indique des usages individuels.

Les tableaux 6.1, 6.2, 6.3, 6.4, 6.5, et 6.6 sont les résultats de l'application de la grille de lecture multiéchelle au scénario que nous venons de présenter. Les notations suivantes sont utilisées :

— vide : lorsque l'échelle n'est pas présente dans le scénario,
— *nd* : Non Détaillé, lorsque l'échelle est (semble) présente dans le scénario mais la description ne donne pas de détail.

TABLE 6.1 – Étude du scénario Ski - Future Internet 2020 pour le point de vue équipement [Arcangeli 2013b]

Scenarios / échelles	**Objets taggés**	**Objets intelligents**	**Équipement personnel**	**Cloudlet**	**Cloud**
Ski - Future Internet 2020	puces RFID	capteurs	mobiles	serveurs dédiés	serveurs

TABLE 6.2 – Étude du scénario Ski - Future Internet 2020 pour le point de vue réseau [Arcangeli 2013b]

Scenarios / échelles	**Réseau personnel**	**Réseau local**	**Réseau métropolitain**	**Internet**
Ski - Future Internet 2020	X	*nd*		X

TABLE 6.3 – Étude du scénario Ski - Future Internet 2020 pour le point de vue donnée [Arcangeli 2013b]

Scenarios / échelles	**Atomiques**	**Structurées**	**BD relationnelles**	**Bases de connaissances**	**Big Data**
Ski - Future Internet 2020	*nd*	X	X		

TABLE 6.4 – Étude du scénario Ski - Future Internet 2020 pour le point de vue géographie [Arcangeli 2013b]

Scenarios / échelles	**Personne**	**Salle**	**Bâtiment**	**Ville**	**Région**	**Monde**
Ski - Future Internet 2020	*nd*	*nd*	*nd*	*nd*	*nd*	*nd*

TABLE 6.5 – Étude du scénario Ski - Future Internet 2020 pour le point de vue utilisateur [Arcangeli 2013b]

Scenarios / échelles	**Individu**	**Groupe**	**Communauté**	**Tous**
Ski - Future Internet 2020	X	X		

TABLE 6.6 – Étude du scénario Ski - Future Internet 2020 pour le point de vue administration [Arcangeli 2013b]

Scenarios / échelles	Individu	Organisation administrative	Inter-organisations
Ski - Future Internet 2020	X	X	

L'étude d'autres scénarios peut être consultée dans le chapitre 2 du livrable *Scénarios et Exigences* du projet INCOME [Arcangeli 2013b].

6.1.2 Extensibilité de la taxonomie et du *framework* MuSCa

Ce travail a permis également d'enrichir la taxonomie MuSCa en y intégrant, scénario par scénario, des points de vue, dimensions, mesures et échelles qui n'y figuraient pas encore. Une représentation de la taxonomie MuSCa au format XML est consultable en annexe A.

Chaque nouveau cas d'utilisation de la plateforme INCOME est caractérisé avec le *framework* MuSCa.

L'utilisation du *framework* MuSCa pour la caractérisation de scénarios multiéchelles permet de mettre en avant l'extensibilité et la facilité d'utilisation comme deux points forts de ce *framework*. En effet, la taxonomie MuSCa permet de faciliter la caractérisation multiéchelle d'un nouveau scénario en aidant l'utilisateur à identifier rapidement les points de vue, dimensions et échelles présentes dans le scénario. De plus, la possibilité d'ajouter des nouveaux concepts à la taxonomie rend le *framework* ouvert et extensible.

Il faut cependant noter que l'ajout d'une nouvelle dimension dans la taxonomie nécessite d'être accompagné du développement d'une sonde basique, ce qui complexifie l'extensibilité du *framework* en ce qui concerne la génération de code pour la conscience des échelles à l'exécution. En revanche, quand une sonde basique est développée pour une dimension, elle est réutilisable pour générer des sondes multiéchelles pour toutes les caractérisations qui utilisent cette dimension, quelles que soient les échelles considérées.

6.2 Déploiement multiéchelle

Le déploiement d'entités logicielles sur des équipements d'un système multiéchelle est une des préoccupations du projet INCOME.

Le déploiement d'un système logiciel réparti est un processus complexe qui a pour objectif la mise à disposition et le maintien en condition opérationnelle du système. Les systèmes multiéchelles sont fortement hétérogènes, dynamiques et ouverts —apparition et disparition d'appareils—. Ils peuvent être composés de centaines de composants logiciels déployés sur des centaines ou des milliers d'appareils. Pour ces systèmes, l'expression du plan de déploiement (association entre les composants logiciels et les machines) ainsi que la réalisation et la supervision du déploiement sont des tâches humainement impossibles du fait de l'hétérogénéité, de la dynamique et de l'ouverture, du nombre, mais aussi parce que le domaine de déploiement (le réseau de machines sur lesquelles on déploie) n'est pas connu à l'avance.

Pour assurer le déploiement, il est nécessaire de disposer d'une part de moyens d'expression et d'abstraction, et d'autre part, de moyens pour la réalisation automatisée (construction du plan, réalisation du plan puis maintien en condition opérationnelle du système).

Les membres du projet INCOME ont recensé les exigences en terme de déploiement sensible au multiéchelle (cf. exigence 28 du tableau 6.7 extrait du livrable [Arcangeli 2013a]). Ces exigences sont décrites dans les mini-scénarios 11 à 16 du livrable [Arcangeli 2013a]. Le mini-scénario 11 (cf. tableau 6.8) est un mini-scénario générique, exprimé pour tout point de vue. Les mini-scénarios 12 à 16 déclinent respectivement ce mini-scénario générique pour les points de vue géographie, utilisateur, administration, équipement et réseau. Nous donnons les mini-scénarios 12 et 13 à titre d'exemples (cf. tableaux 6.9 et 6.10).

TABLE 6.7 – Exigence INCOME : déploiement conscient des échelles [Arcangeli 2013a]

Requirement 28: Multiscalability-aware deployment of the context management entities.
During the deployment phase, the choice of the appropriate deployment strategy SHOULD take into account the multiscale aspects at the geographical, network, device and user dimensions. Those choices may also take into account extra-functional properties such as efficiency and privacy.

TABLE 6.8 – Mini-scénario INCOME : expression de contraintes de déploiement multiéchelles [Arcangeli 2013a]

Mini-scenario 11: Multiscalability, expression of a deployment constraint on a top viewpoint.
With the deployment DSL, a deployment designer expresses a deployment constraint concerning this viewpoint; she/he wants to specify a constraint on the device(s) on which the context management entities will be deployed, the constraint concerns the viewpoints scale(s) to which the devices should belong.

TABLE 6.9 – Mini-scénario INCOME : expression de contraintes de déploiement multiéchelles, point de vue géographie [Arcangeli 2013a]

Mini-scenario 12: Multiscalability, expression of a deployment constraint, viewpoint "geography", geography scale requirement.
With the deployment DSL, a deployment designer expresses a deployment constraint concerning the geography viewpoint; she/he wants to specify a constraint on the device(s) on which the context management entities will be deployed; the constraint concerns the geography scale(s) the devices should belong to.

TABLE 6.10 – Mini-scénario INCOME : expression de contraintes de déploiement multiéchelles, point de vue utilisateur [Arcangeli 2013a]

Mini-scenario 13: Multiscalability, expression of a deployment constraint, viewpoint "user", user scale requirement.
With the deployment DSL, a deployment designer expresses a deployment constraint concerning the user viewpoint; she/he wants to specify a constraint on the device(s) on which the context management entities will be deployed; the constraint concerns the user scale(s) the devices should belong to.

Tout d'abord, afin de répondre aux besoins d'expression et d'abstraction de plans de déploiement pour les systèmes répartis multiéchelles, les membres du projet INCOME ont défini un langage dédié (Domain Specific Language, DSL), appelé MuScADeL [Boujbel 2013], consacré au déploiement logiciel autonomique et multiéchelle. Ce langage intègre directement les principaux concepts du métamodèle MuSCa : point de vue, dimension et échelle. Ainsi, MuScADeL permet aux concepteurs du déploiement de déclarer des propriétés de déploiement de façon abstraite, sans avoir une connaissance exacte des équipements et des réseaux sur lesquels le système sera déployé. De plus, grâce à l'intégration de MuSCa dans MuScADeL, les propriétés de déploiement peuvent être liées à une caractériation MuSCa du système à déployer.

Un exemple d'utilisation du langage MuScADeL, appliqué au déploiement du scénario décrit dans la section 1.4, est présenté dans la figure 6.1. Les lignes 4 à 6 servent à déclarer l'existence des sondes multiéchelles pour les points de vue réseau, géographie et équipement, afin de pouvoir y faire référence dans la suite du programme. La ligne 8 définit un critère concernant la bande passante minimale d'un équipement. Les lignes 10 à 23 définissent les propriétés de déploiement exprimées dans le scénario de la section 1.4. Ces propriétés concernent trois types de composants. Tout d'abord, la ligne 12 exprime qu'un composant dédié au filtrage doit être installé dans chaque réseau LAN impliqué dans le système, sur une machine possédant une bande passante supérieure à $50MB/s$. Puis, les lignes 14 à 19 décrivent les propriétés de déploiement hiérarchique des composants de routage : un dans chaque immeuble de *Toulouse*, un dans chaque quartier, et un pour la ville. Ce dernier doit de plus être déployé sur une machine associée à l'échelle *Cloud* pour la dimension puissance de calcul du point de vue équipement. Enfin, les lignes 21 et 22 expriment que les composants conscients du contexte, destinés aux utilisateurs finaux, doivent être déployés sur tous les *smartphones* de la ville de *Toulouse*.

Les sondes multiéchelles déclarées sont déployées sur l'ensemble du domaine de déploiement. Les valeurs retournées par les sondes sont utilisées pour construire un plan de déploiement conforme aux propriétés définies. Ce plan contient l'ensemble des instructions de déploiement pour chaque composant du système.

L'éditeur MuScADeL fonctionne, tout comme l'éditeur MuSCa, sous forme d'un *plugin* Eclipse, ce qui permet aux concepteurs du déploiement d'utiliser le même environnement de développement (Eclipse) pour éventuellement définir de nouveaux points

```
2  // Definition of probes
3  Probe Network {...}
4  MultiScaleProbe MSNetwork {..}
5  MultiScaleProbe Geography {..}
6  MultiScaleProbe Device {..}
7  // Definition of a criterion
8  BCriterion Crit50MB { Network.bandwidth > 50; }
9  // Definition of deployment requirements
10 Deployment {
11         // deployment of filter components
12         F @ Crit50Mb, Each MSNetwork.Type.LAN;
13         // deployment of hierarchical routing components
14         R @ Each Geography.Location.Building,
15             Geography.Location.City("Toulouse");
16         R @ Each Geography.Location.District,
17             Geography.Location.City("Toulouse");
18         R @ Geography.Location.City("Toulouse"),
19             Device.ProcessingPower.Cloud;
20         // deployment of end-user context-aware components
21         C @ All Device.Type.Smartphone,
22             Geography.Location.City("Toulouse");
23 }
```

FIGURE 6.1 – Contraintes de déploiement multiéchelle

de vues, dimensions ou échelles, afin de les utiliser dans le DSL de déploiement. Étant donné que le code MuScADeL est lié à un modèle de caractérisation MuSCa, l'éditeur MuScADeL peut vérifier que les points de vue, dimensions et échelles utilisés correspondent à ceux utilisés dans la caractérisation MuSCa associée. De plus, comme le montre la figure 6.2, l'éditeur propose un mécanisme de complétion automatique qui permet d'écrire rapidement des contraintes de déploiement à partir des éléments contenus dans la caractérisation MuSCa associée.

Dans le cadre de ce travail, les concepts du métamodèle MuSCa ont facilité l'expression de contraintes de déploiement multiéchelles. Cependant, il est à noter que le langage MuScADeL n'intègre pas l'ensemble des concepts de MuSCa. En particulier, le concept de mesure n'a pas été conservé. Les concepteurs de MuScADeL ont posé comme hypothèse de base de ne garder qu'une mesure par dimension. Cette hypothèse permet de simplifier la syntaxe du langage en exprimant une échelle sous la forme `Viewpoint.Dimension.Scale` au lieu de `Viewpoint.Dimension.Measure.Scale`. Cette hypothèse est acceptable car nous ne voyons pas de cas d'utilisation qui nécessiterait la cohabitation de deux ensembles d'échelles —associés à des mesures différentes— pour une même dimension.

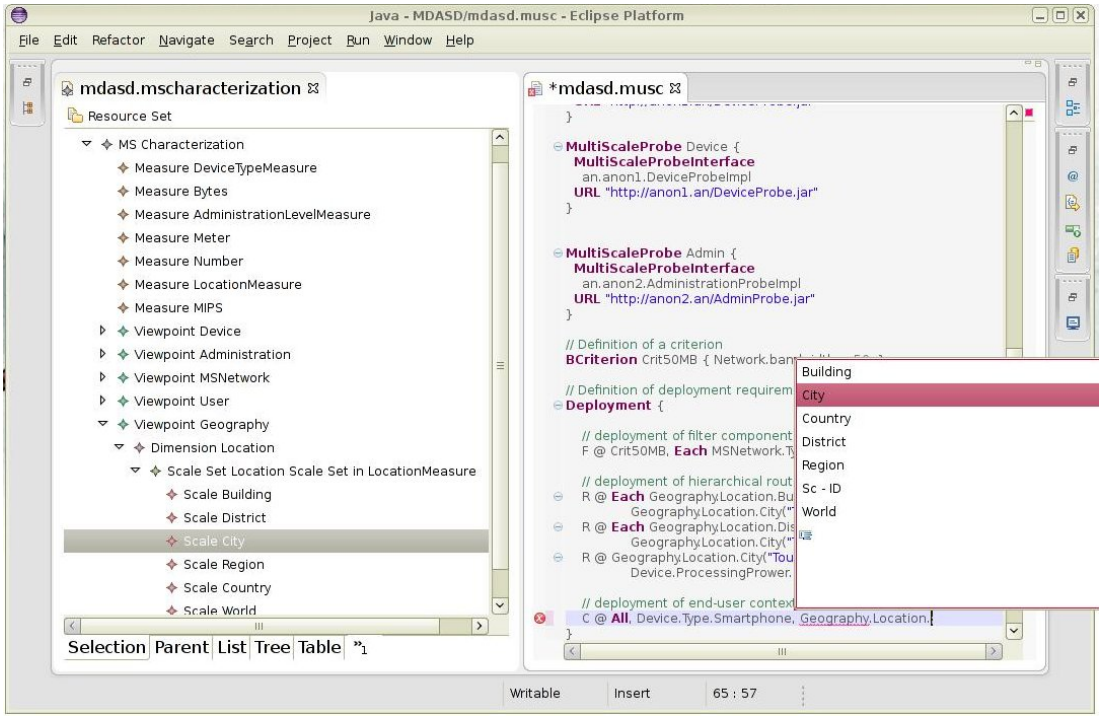

FIGURE 6.2 – Complétion automatique de l'éditeur MuScADeL à partir d'un modèle MuSCa

Par ailleurs, des travaux ont été réalisés concernant la réalisation automatisée du déploiement. Cette réalisation consiste notamment à vérifier que les propriétés de déploiement sont respectées à tout moment au cours de l'exécution du système. Dans ce but, les sondes multiéchelles générées à partir de la caractérisation MuSCa sont exécutées pour la construction du plan de déploiement. Ces sondes sont déployées sur l'ensemble des équipements du domaine de déploiement. Elles sont utilisées pour superviser l'état de ces équipements en termes d'échelles. Elles permettent de choisir les machines sur lesquelles sont exécutées les composants en fonction des contraintes multiéchelles exprimées. En appelant régulièrement ces sondes, il est également possible de détecter un changement d'échelle pour l'un des équipements, et dans le cas où une contrainte ne serait plus respectée, de modifier le déploiement du système.

6.3 Gestion de la diffusion d'informations de contexte

La gestion de la diffusion d'informations de contexte au sein d'un gestionnaire de contexte réparti fait également l'objet de travaux dans le cadre du projet ANR INCOME. Le *framework* MuSCa est utilisé dans ces travaux afin d'y ajouter une composante multiéchelle. Les membres du projet INCOME étudient la gestion de la diffusion d'informations de contexte à travers trois solutions, qui correspondent aux trois parties de cette section : la définition de contrats de production et de consommation d'informations de contexte (section 6.3.1), le filtrage des notifications (section 6.3.2), et le routage des notifications (section 6.3.3).

6.3.1 Définition de contrats de production et de consommation d'informations de contexte

Un gestionnaire de contexte met en relation des producteurs et des consommateurs d'informations de contexte. Les contrats de production et de consommation permettent aux producteurs et aux consommateurs de spécifier leurs exigences et leurs garanties en termes de confidentialité et de qualité de contexte [Machara Marquez 2013]. Cependant, il est nécessaire de compléter le langage de définition de contrats afin d'y intégrer des contraintes multiéchelles. Ce besoin est exprimé dans les mini-scénarios 3 à 6 (cf. tableaux 6.11, 6.12, 6.13 et 6.14), extraits du livrable [Arcangeli 2013a]. Le mini-scénario 3 est un mini-scénario générique, exprimé pour tout point de vue. Les mini-scénarios 4 à 6 déclinent respectivement ce mini-scénario générique pour les points de vue géographie, utilisateur et administration.

TABLE 6.11 – Mini-scénario INCOME : expression de contraintes multiéchelles dans un contrat de contexte [Arcangeli 2013a]

Mini-scenario 3: Multiscalability, expression of context contract, any viewpoint, viewpoint scale requirement.
A context contract designer uses the context contract DSL to express a context contract constraint on a given viewpoint; she/he specifies constraints concerning the viewpoint scale(s) to which the projections of the context consumer and of the observable entity onto this viewpoint must belong.

Afin d'adapter les contrats de contexte à un gestionnaire de contexte multiéchelle,

TABLE 6.12 – Mini-scénario INCOME : expression de contraintes multiéchelles dans un contrat de contexte, point de vue géographie [Arcangeli 2013a]

Mini-scenario 4: Multiscalability, expression of context contract, viewpoint "geography", geography scale requirement.
A context contract designer uses the context contract DSL to express a context contract constraint on the geography viewpoint; she/he specifies a constraint on the geography scale to which the projections of the context consumer and of the observable entity onto the geography viewpoint must belong.

TABLE 6.13 – Mini-scénario INCOME : expression de contraintes multiéchelles dans un contrat de contexte, point de vue utilisateur [Arcangeli 2013a]

Mini-scenario 5: Multiscalability, expression of context contract, viewpoint "user", user scale requirement.
A context contract designer uses the context contract DSL to express a context contract constraint on the user viewpoint; she/he specifies constraints concerning the user scale(s) to which the projections of the context consumer and of the observable entity onto the user viewpoint must belong.

TABLE 6.14 – Mini-scénario INCOME : expression de contraintes multiéchelles dans un contrat de contexte, point de vue administration [Arcangeli 2013a]

Mini-scenario 6: Multiscalability, expression of context contract, viewpoint "administration", administration scale requirement.
A context contract designer uses the context contract DSL to express a context contract constraint on the administration viewpoint; she/he specifies constraints concerning the administration scale(s) to which the projections of the context consumer and of the observable entity onto the administration viewpoint must belong.

des travaux sont en cours pour permettre d'exprimer des exigences et des garanties relatives aux échelles dans ces contrats. L'objectif de ces travaux est d'exprimer des exigences ou des garanties de confidentialité ou de qualité de contexte à partir d'éléments d'une caractérisation multiéchelle, c'est-à-dire d'un modèle MuSCa. Par exemple, un consommateur en voiture, arrivé à destination, serait capable d'exprimer qu'il souhaite obtenir les informations provenant de places de stationnement situées à une distance accessible à pied autour de lui. Cela correspond à l'expression d'une contrainte relative

à l'instance d'échelle « accessible à pied » pour le couple géographie/distance (point de vue/dimension) de ce consommateur. Dans un deuxième exemple, un producteur pourrait exprimer qu'il souhaite uniquement partager ses informations de contexte avec les utilisateurs situés dans son quartier. Cela correspond à l'expression d'une contrainte relative à l'instance d'échelle « quartier » pour le couple géographie/localisation administrative de ce consommateur. Ces contraintes sont définies spécifiquement à partir des éléments présents dans le modèle MuSCa de caractérisation multiéchelle du système en cours d'exécution. Ainsi, les concepteurs de contrats de contexte ont accès à un vocabulaire à la fois commun à l'ensemble du système et restreint aux échelles identifiées pour ce système.

6.3.2 Filtrage des informations de contexte et conscience des échelles

Les contraintes définies dans les contrats doivent ensuite être appliquées à l'exécution pour contrôler la diffusion des informations de contexte au sein du gestionnaire de contexte. Ce besoin est exprimé dans les mini-scénarios 7 à 10 (cf. tableaux 6.15, 6.16, 6.17 et 6.18), extraits du livrable [Arcangeli 2013a]. Comme précédemment, le mini-scénario 7 est un mini-scénario générique, exprimé pour tout point de vue. Les mini-scénarios 8 à 10 déclinent respectivement ce mini-scénario générique pour les points de vue géographie, utilisateur et administration.

TABLE 6.15 – Mini-scénario INCOME : application de contraintes multiéchelles provenant de contrats de contexte [Arcangeli 2013a]

Mini-scenario 7: Multiscalability: enforcement by the context management service of a viewpoint scale requirement defined in a context contract.
A context consumer (an end-user application or a context capsule) consumes context data, concerning an observable entity, according to a context consumer contract and a context producer contract. On a given viewpoint, each context contract may specify a constraint concerning the viewpoint scale(s) to which the projections of the context consumer and of the observable entity onto this viewpoint must belong. The context management service insures that this constraint is fulfilled.

La solution choisie par les membres du projet INCOME est la mise en place de filtres de consommation et de production. Les filtres sont appliqués aux informations

TABLE 6.16 – Mini-scénario INCOME : application de contraintes multiéchelles provenant de contrats de contexte, point de vue géographie [Arcangeli 2013a]

Mini-scenario 8: Multiscalability, distributed architecture, viewpoint "geography", measure "distance between the consumer and the observable entity"; **Multiscalability: enforcement by the context management service of a geography scale requirement defined in a context contract.**
A context consumer (an end-user application or a context capsule) consumes context data, concerning an observable entity, according to a context consumer contract and a context producer contract. The context contracts specify constraints concerning the distance between the consumer and the observable entity. The context management service insures that these constraints are fulfilled.

TABLE 6.17 – Mini-scénario INCOME : application de contraintes multiéchelles provenant de contrats de contexte, point de vue utilisateur [Arcangeli 2013a]

Mini-scenario 9: Multiscalability, distributed architecture, viewpoint "user", measure "category of user group" **Multiscalability: enforcement by the context management service of a user scale requirement defined in a context contract.**
A context consumer (an end-user application or a context capsule) consumes context data, concerning an observable entity, according to a context consumer contract and a context producer contract. On the user viewpoint, each context contract may specify a constraint concerning the user scale(s) to which the projections of the context consumer and of the observable entity onto the user viewpoint must belong. The context management service insures that this constraint is fulfilled.

de contexte, appelées notifications, qui circulent dans le gestionnaire de contexte. Ils permettent d'évaluer si une notification doit être livrée ou non à un consommateur.

TABLE 6.18 – Mini-scénario INCOME : application de contraintes multiéchelles provenant de contrats de contexte, point de vue administration [Arcangeli 2013a]

Mini-scenario 10: Multiscalability, distributed architecture, viewpoint "administration", measure category of administrative organization; **Multiscalability: enforcement by the context management service of an administration scale requirement defined in a context contract.**
A context consumer (an end-user application or a context capsule) consumes context data, concerning an observable entity, according to a context consumer contract and a context producer contract. On the administration viewpoint, each context contract may specify a constraint concerning the administration scale(s) to which the projections of the context consumer and of the observable entity onto the administration viewpoint must belong. The context management service insures that this constraint is fulfilled.

6.3.2.1 Rapport multiéchelle

Afin de pouvoir évaluer le respect des contraintes liées au multiéchelle, il est nécessaire d'ajouter aux notifications des méta-informations de conscience des échelles. Pour une notification donnée, ces méta-informations sont contenues dans ce que nous appelons un rapport multiéchelle. Le rapport multiéchelle d'une notification contient la caractérisation multiéchelle du producteur de la notification, c'est-à-dire les projections dans les points de vue et éventuellement les échelles associées au producteur, au moment où il produit la notification. Un tel rapport est produit à l'exécution, au moment de la production d'une notification, à l'aide des sondes multiéchelles générées à partir du modèle MuSCa utilisé pour la spécification des contrats. Un exemple de rapport multiéchelle est présenté dans la figure 6.3. Ce rapport a été produit à l'aide de sondes multiéchelles pour les points de vue équipement et géographie. La sonde multiéchelle géographique a été implémentée par des appels à une sonde de bas niveau qui utilise l'API de *reverse geocoding* du service Nominatim [3] de OpenStreetMap [4]. Un rapport multiéchelle contient une partie par point de vue présent dans la caractérisation MuSCa du système en cours d'exécution : dans la figure 6.3 ce sont les points

3. http://wiki.openstreetmap.org/wiki/Nominatim
4. http://www.openstreetmap.org

de vue géographie et équipement. Pour le point de vue géographie, les lignes 7 à 11 du rapport représentent la projection de l'entité sur ce point de vue, c'est-à-dire un identifiant unique ainsi que les coordonnées de géolocalisation (latitude et longitude) de l'entité. Les lignes 12 à 41 contiennent les échelles et instances d'échelles associées à l'entité pour la dimension localisation administrative. De même, pour le point de vue équipement, les lignes 44 à 46 représentent la projection de l'entité sur ce point de vue, c'est-à-dire un identifiant unique. La ligne 47 présente l'échelle à laquelle l'entité est associée pour la dimension capacité de stockage.

Ce rapport indique que la notification à laquelle il est associé a été produite à *Toulouse*, dans le quartier du *Capitole*, depuis un équipement associé à l'échelle *giga* pour la dimension capacité de stockage.

Remarque. *Comme nous pouvons le constater sur la figure 6.3, la structure d'un rapport multiéchelle dépend d'une caractériation MuSCa, c'est-à-dire qu'un rapport contient les projections, échelles et instances d'échelles associées au producteur pour les points de vue et dimensions présentes dans le modèle MuSCa du système en cours d'exécution. Le code à générer à partir d'une caractérisation est présenté en annexe C. Nous travaillons actuellement sur l'implémentation d'un générateur de code pour produire automatiquement des artefacts logiciels dédiés à la création de rapports multiéchelles à partir d'un modèle MuSCa.*

Le rapport multiéchelle contient au minimum les projections associées au producteur pour les points de vue choisis par la caractérisation. Les échelles peuvent au choix être intégrées dans les rapports multiéchelles ou recalculées à partir des projections pendant l'exécution d'un filtre. Le choix de restreindre les points de vue, dimensions et échelles présents dans un rapport multiéchelle aux éléments présents dans la caractérisation MuSCa du modèle en cours d'exécution permet de réduire le volume d'un tel rapport au strict minimum. Cette solution est préférable à une autre solution qui aurait consisté à intégrer systématiquement tous les éléments de la taxonomie MuSCa.

6.3.2.2 Filtres

Au moment de décider si une notification doit être livrée ou non à un consommateur, des filtres sont exécutés pour vérifier si les contraintes des contrats de contexte sont

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<muSCaMultiscalabilityReport>
    <id>report</id>
    <projectionvalues/>
    <reports/>
    <geographyViewpointReport>
        <geographyProjection>
            <id>localid</id>
            <latitude>43.602905</latitude>
            <longitude>1.442682</longitude>
        </geographyProjection>
        <location_In_OSMNominatimLocationRank_ScaleTree>
            <entry>
                <key>POINT</key>
                <value>43.602905;1.442682</value>
            </entry>
            <entry>
                <key>NEIGHBORHOOD</key>
                <value>Capitole</value>
            </entry>
            <entry>
                <key>SUBURB</key>
                <value>Toulouse Centre</value>
            </entry>
            <entry>
                <key>CITY</key>
                <value>Toulouse</value>
            </entry>
            <entry>
                <key>REGION</key>
                <value>Midi-Pyrénées</value>
            </entry>
            <entry>
                <key>COUNTRY</key>
                <value>fr</value>
            </entry>
            <entry>
                <key>WORLD</key>
                <value>World</value>
            </entry>
        </location_In_OSMNominatimLocationRank_ScaleTree>
    </geographyViewpointReport>
    <deviceViewpointReport>
        <deviceProjection>
            <id>deviceid</id>
        </deviceProjection>
        <storageCapacity_In_Bytes_Scale>GIGA</storageCapacity_In_Bytes_Scale>
    </deviceViewpointReport>
</muSCaMultiscalabilityReport>
```

FIGURE 6.3 – Exemple de rapport multiéchelle

respectées. Ces filtres utilisent notamment les informations contenues dans le rapport multiéchelle de la notification. Ils sont également implémentés à l'aide des sondes multiéchelles générées grâce à l'approche IDM. Dans le cadre des filtres, ces sondes peuvent

être utilisées pour caractériser le consommateur de façon multiéchelle. Le filtre peut alors évaluer si le rapport multiéchelle de la notification respecte ou non les contraintes des contrats de contexte.

Un filtre peut être défini par un consommateur puis transporté du côté producteur. Celui-ci peut vérifier si une information doit être diffusée ou non vers le consommateur. À l'aide du filtre présenté dans la figure 6.4, le consommateur demande à ce que seules les informations concernant la ville de *Toulouse* lui soient transmises.

```
1  private String filterAtester = "function evaluate(doc) {"
2                + "importPackage(javax.xml.xpath);"
3                + "var i,instance;"
4                + "var ville='Toulouse';"
5                + "var xpath = XPathFactory.newInstance().newXPath();"
6                + "var scales = xpath.evaluate('//muSCaMultiscalabilityReport/
                     geographyViewpointReport/
                     location_In_OSMNominatimLocationRank_ScaleTree/entry/key',
                     doc, XPathConstants.NODESET);"
7                + "  for (i = 0; i < scales.getLength(); i++) {"
8                      + "if (scales.item(i).getTextContent().equalsIgnoreCase('
                         CITY')) {"
9                      + "    instance = xpath.evaluate('../value', nodes.item(i
                         ), XPathConstants.NODE);"
10                       +"    if (instance.getTextContent().
                             equalsIgnoreCase(ville)) return true;"
11                     + "}"
12               + "  }"
13               + "return false;}";
```

FIGURE 6.4 – Exemple de filtre d'information de contexte conscient des échelles

À titre d'exemple, la figure 6.5 illustre les zones qui correspondent aux contraintes de filtrage de données de contexte présentées dans le scénario de la section 1.4, c'est-à-dire les instances des échelles *quartier* (en bleu pour la dimension *localisation administrative*) et *accessible à pied* (en jaune pour la dimension *distance*) de l'utilisatrice Sophie au moment de l'exécution du filtre.

La figure 6.5 a été produite grâce à une librairie logicielle que nous avons développée au-dessus de l'API JMapViewer [5]. Cette librairie permet de visualiser simplement les instances d'échelles associées aux entités d'un système.

5. http://wiki.openstreetmap.org/wiki/JMapViewer

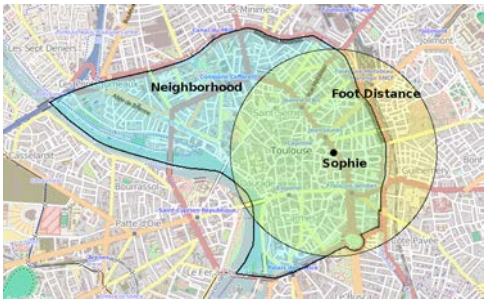

FIGURE 6.5 – Les instances d'échelles géographiques de Sophie

6.3.3 Routage des informations de contexte et *scoping* conscient des échelles

D'autres travaux du projet INCOME concernent le routage des informations de contexte. Le routage consiste à contrôler le chemin que parcourt une donnée de contexte. Ces travaux viennent s'ajouter aux travaux effectués sur le filtrage des données, qui permettent de savoir si un client peut recevoir ou non une donnée. L'approche étudiée pour le routage des données de contexte est le *multiscoping* [Conan 2014]. Le *scoping* consiste à regrouper les clients du gestionnaire de contexte (producteurs et consommateurs) en *scopes*. Un *scope* définit un périmètre de visibilité, c'est-à-dire un ensemble de clients pouvant communiquer entre eux. Les *scopes* sont organisés en graphes de *scopes* à l'aide de la relation *super-scope*, et de la relation réciproque *sous-scope*. Un *super-scope* est constitué de l'ensemble des clients contenus dans ses *sous-scopes*. Dans la figure 6.6, les *scopes* T et U ont pour *super-scope* S. Réciproquement, S a pour *sous-scopes* T et U.

Nous proposons de faire correspondre cette organisation à une caractérisation multiéchelle afin de définir des périmètres de visibilité basés sur les différentes échelles et instances d'échelles du système. En particulier, nous proposons de construire un graphe de *scopes* par ensemble d'échelles, c'est-à-dire pour un couple dimension/mesure donné. Les *scopes* étant naturellement organisés sous forme de graphes, nous décidons de ne considérer que les ensembles d'échelles de dimensions hiérarchiques (cf. section 4.1.3). En effet, une organisation hiérarchique est naturellement représentable par un graphe, et plus précisément par un arbre. Par exemple, pour une caractérisation qui contiendrait le point de vue géographie et la dimension localisation administrative (avec sa mesure sémantique associée), il est possible de construire le graphe de scopes —ou l'arbre— de la figure 6.7. Chaque *scope* représente une instance d'échelle, et son niveau hiérar-

chique dans l'arbre correspond à une échelle. Par exemple, le *scope* Toulouse représente l'instance d'échelle Toulouse (niveau M0) de l'échelle Ville (niveau M1).

Cependant, toutes les dimensions hiérarchiques ne sont pas adaptées à la construction d'un graphe de *scopes*. En effet, nous pouvons remarquer qu'il existe deux types de dimensions hiérarchiques, que nous appelons relatives ou absolues. Par exemple, pour le point de vue géographie (cf. figure 4.4), la dimension distance implique la définition d'échelles dont les instances dépendent d'une entité donnée : accessible à pied depuis l'entité, accessible en voiture depuis l'entité, etc. Cette dimension est donc appelée relative, car les instances d'échelles définies pour cette dimension sont relatives à une entité. À l'inverse, la dimension localisation administrative est absolue car elle implique la définition d'échelles dont les instances sont indépendantes des entités du système : une ville, un pays, etc. Si l'on souhaitait construire un graphe de *scopes* pour une dimension relative, cela impliquerait de construire un graphe de *scopes* pour chaque entité du système, ce qui n'est pas une solution acceptable car elle ne passe pas à l'échelle. Ainsi, seules les dimensions hiérarchiques absolues sont considérées pour la création d'un graphe de *scopes*.

Un graphe de *scopes* peut être construit dynamiquement lors du déploiement des entités. À cet instant, les instances d'échelles effectivement présentes dans le système sont connues.

Le *framework* MuSCa a permis de formaliser la notion de *multiscoping*. Le concept d'échelle est associé au concept de *scope* (portée). La variété des points de vue et dimensions permet de définir du *multiscoping* pour exprimer que la portée d'une notification est contrainte selon plusieurs critères, c'est-à-dire dans plusieurs zones logiques correspondant chacune à une instance d'échelle dans une dimension. Les clients (producteurs et consommateurs) estampillent leurs filtres d'annonce et de souscription avec une échelle par dimension. Une notification d'un producteur est visible pour un consommateur si elle est visible dans toutes les dimensions.

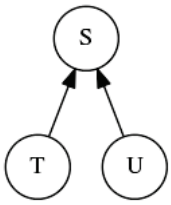

FIGURE 6.6 – *Scoping*

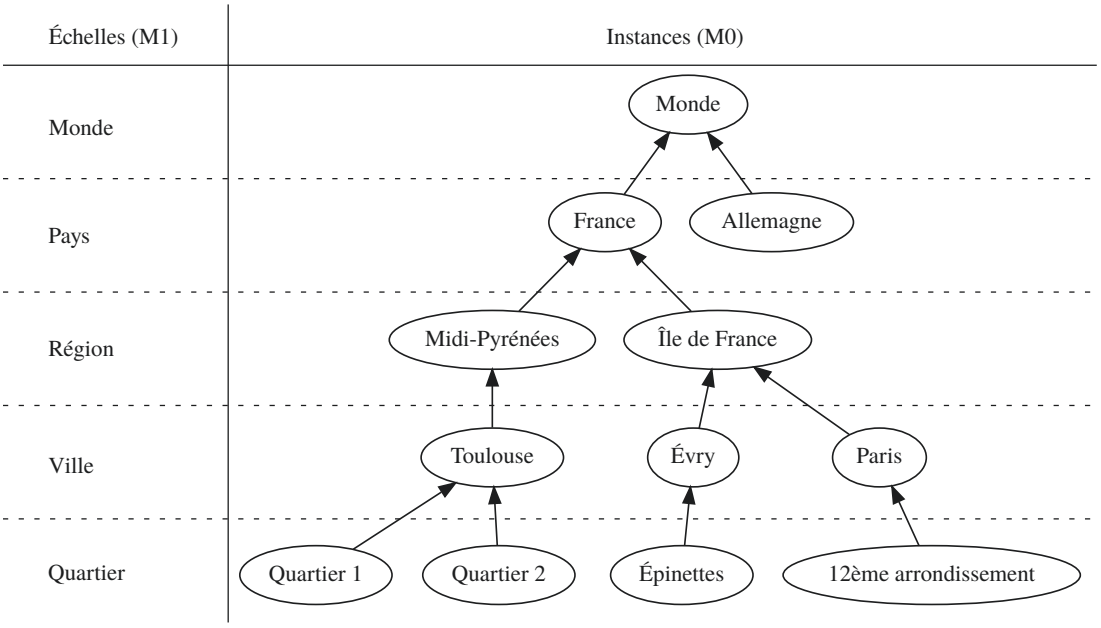

FIGURE 6.7 – Graphe de *scopes* : Géographie / Localisation administrative

6.4 Bilan

Dans ce chapitre, nous avons validé qualitativement notre proposition suivant deux axes : d'une part l'expressivité du vocabulaire et du métamodèle MuSCa ainsi que l'apport de la taxonomie, et d'autre part la démarche IDM avec génération de sondes multiéchelles. Pour cela, nous avons étudié trois travaux de recherche effectués dans le cadre du projet ANR INCOME. L'étude de ces travaux montre que le *framework* MuSCa répond aux principaux besoins du projet concernant les problématiques du multiéchelle.

D'une part, MuSCa apporte des concepts ainsi qu'une taxonomie du multiéchelle

qui peuvent être intégrés dans différents langages dédiés. Ceci a été mis en œuvre avec succès pour la spécification de plans de déploiement avec le langage MuScADeL, et des travaux sont en cours pour faire de même dans le cadre de la définition de contrats de production et de consommation d'informations de contexte.

D'autre part, les sondes multiéchelles générées à partir d'une caractérisation MuSCa peuvent être utilisées pour implémenter des mécanismes de vérification des différentes contraintes qui ont été exprimées à l'aide des langages dédiés.

Nous pouvons constater que le *framework* MuSCa est suffisamment générique pour pouvoir être utilisé dans des domaines variés. L'apport scientifique de ce *framework* est en particulier validé par son utilisation dans différentes tâches du projet ANR INCOME. Il a répondu aux exigences scientifiques du projet relatives au multiéchelle. Le *framework* MuSCa apporte une expressivité efficace et simplifiée. Les contraintes sont spécifiées de manière déclarative pour des combinaisons de points de vue dimensions et échelles, et sont donc valables pour des ensembles d'entités. De plus, le *framework* permet de s'adapter spécifiquement à un système donné, car les éléments utilisés dans la spécification des contraintes proviennent d'une caractérisation MuSCa propre à ce système. Enfin, l'approche IDM permet d'utiliser le *framework* pour implémenter des mécanismes d'application des contraintes à l'exécution, à l'aide d'artefacts logiciels générés spécifiquement pour une caractérisation MuSCa donnée. Actuellement, les artefacts générés automatiquement sont les sondes multiéchelles. Nous travaillons sur des extensions du *framework* afin de générer, par exemple, des artefacts de production de rapports multiéchelles pour le filtrage de données de contexte.

Conclusions et Perspectives

Chapitre 7

Conclusions et Perspectives

7.1 Conclusions

Cette thèse avait pour objectif de répondre à la problématique de simplification des étapes de conception, développement et déploiement de systèmes répartis complexes que l'on peut qualifier de multiéchelles, en proposant des méthodes et des outils aussi génériques que possible. En effet, ces tâches sont aujourd'hui fastidieuses et reposent sur des solutions *ad hoc* peu réutilisables. Nous avons pu montrer que l'application d'une vision multiéchelle à ces systèmes permet d'en abstraire une organisation plus simple. Nous avons adopté une approche descendante basée sur l'IDM.

Nous avons proposé, dans le chapitre 4, un vocabulaire et un processus de caractérisation multiéchelle pour les systèmes répartis complexes. Ce processus permet d'identifier précisément les différents points de vue, dimensions et échelles présentes dans un système réparti complexe donné, afin d'en qualifier la nature multiéchelle. Nous avons précisé et formalisé ce processus à l'aide de concepts mathématiques de la théorie des ensembles.

Puis, dans le chapitre 5, nous avons mis en œuvre le processus de caractérisation multiéchelle en proposant le *framework* MuSCa. Ce dernier permet à un concepteur de caractériser la nature multiéchelle du système qu'il souhaite concevoir. Pour cela, il a à sa disposition une taxonomie qui contient tous les éléments de caractérisation identifiés au cours de précédentes caractérisations. Cette taxonomie est extensible, ce qui permet au concepteur d'y ajouter d'éventuels nouveaux éléments de caractérisation (nouveaux points de vue, dimensions, échelles, etc.). Le résultat de la phase de caractérisation d'un système est exprimé sous la forme d'un modèle de caractérisation multiéchelle. Afin de pouvoir exploiter la caractérisation multiéchelle d'un système au cours de son

exécution, nous proposons, dans le *framework* MuSCa, un mécanisme de génération d'artefacts pour la conscience des échelles, appelés sondes multiéchelles. Ce mécanisme de génération de code repose sur les principes de l'IDM. Un modèle de caractérisation multiéchelle d'un système donné est utilisé en entrée de la génération, ce qui permet de produire des sondes multiéchelles adaptées spécifiquement à cette caractérisation. Grâce aux sondes multiéchelles, il est possible de savoir à tout moment, à l'exécution, à quelle(s) échelle(s) sont associés les entités ou ensembles d'entités du système. Le *framework* MuSCa est implémenté et fonctionnel. Il s'installe sous la forme de *plugins* pour l'environnement de développement Eclipse.

Enfin, dans le chapitre 6, nous avons validé qualitativement notre démarche à travers trois cas d'utilisation réalisés dans le cadre du projet ANR INCOME. Le premier cas d'utilisation concernait la caractérisation multiéchelle de différents scénarios de l'Internet des objets. Il a permis de valider d'une part l'utilisation de la caractérisation multiéchelle comme une grille de lecture pour ces scénarios, et d'autre part l'extensibilité de la taxonomie MuSCa. Le deuxième cas d'utilisation portait sur le déploiement multiéchelle, et le troisième concernait le filtrage et le routage d'informations de contexte au sein d'un gestionnaire de contexte multiéchelle. Ces deux cas d'utilisation ont permis de valider d'une part l'expressivité du vocabulaire multiéchelle, à travers la définition de contraintes, et d'autre part l'utilisation des sondes multiéchelles générées pour l'application de ces contraintes. Ces travaux de validation ont montré la concordance des contributions de cette thèse avec les exigences du projet INCOME. Les travaux réalisés dans le cadre de la thèse, et en particulier l'implémentation du *framework* MuSCa, sont aujourd'hui entièrement intégrés au projet INCOME.

7.2 Perspectives

Les travaux réalisés au cours de cette thèse permettent des prolongements dans le domaine de l'ingénierie et ouvrent des perspectives de recherche.

Nous proposons tout d'abord deux prolongements liés à des travaux d'ingénierie. Le premier consiste à continuer le développement de sondes de bas niveau afin de pouvoir générer, de façon automatique, des sondes multiéchelles pour un plus grand nombre de dimensions. Le second prolongement consiste à faire évoluer les mécanismes de génération de sondes multiéchelles en y intégrant des étapes de transformations de modèles

(*Model to Model*, *M2M*). Cette amélioration permettrait de rendre le mécanisme de génération de code plus maintenable et évolutif.

Cette thèse ouvre également la voie à des perspectives de recherche directement liées aux travaux réalisés. Nous en détaillons deux.

La première perspective de recherche vise à construire une base de technologies réutilisables organisées à l'aide des concepts du vocabulaire multiéchelle. L'objectif serait de pouvoir consolider une bibliothèque ouverte et extensible de *middleware* qui pourraient être utilisés au cours de la conception d'un système réparti complexe, en fonction des échelles identifiées dans la caractérisation multiéchelle de ce système. Par exemple, le choix d'un protocole de communication entre deux entités peut dépendre des échelles respectives auxquelles sont associées ces entités. Une approche envisageable pourrait être de répertorier des *middleware* existants et adaptés aux communications intra-échelle ou inter-échelles. À chaque échelle présente dans la taxonomie multiéchelle pourraient être associés des *middleware* adaptés pour les interactions au sein de cette échelle. De même, à un couple d'échelles (d'un même ensemble d'échelles) pourraient être associés des *middleware* adaptés aux interactions entre les deux échelles de ce couple. Au-delà de la phase de conception, cette approche pourrait être étendue en exploitant les mécanismes de conscience des échelles à l'exécution d'un système afin d'en adapter le comportement. Il serait alors possible de choisir dynamiquement les *middleware* à utiliser, parmi ceux présents dans la base de technologie, en fonction des échelles identifiées à l'exécution par les mécanismes de conscience des échelles.

La seconde perspective de recherche consiste à apporter une dimension sémantique au processus de caractérisation et à la taxonomie multiéchelle en étudiant les liens entre les différents éléments de caractérisation multiéchelle. En effet, il existe des liens immédiats entre certains points de vue multiéchelles. Par exemple, les points de vue réseau et géographie sont intimement liés. Il est en effet possible de considérer que la présence de l'échelle *pays* du point de vue géographie implique la présence de l'échelle *WAN* pour le point de vue réseau dans la même caractérisation. Cette approche viserait à transformer la taxonomie multiéchelle en une ontologie qui définirait les liens sémantiques existants entre les différents éléments de caractérisation multiéchelle. Une telle approche permettrait de mieux guider le concepteur d'un système pendant la caractérisation multiéchelle d'un système. L'éditeur de caractérisation pourrait, par exemple, inférer une caractérisation complète à partir d'une caractérisation simplifiée entrée par

le concepteur, en y ajoutant les points de vue, dimensions et échelles déduites.

Annexes

Annexe A

Taxonomie MuSCa

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This file is part of the MuSCa framework.

Copyright (c) 2011-2015 Institut Mines-Télécom / Télécom SudParis.

The MuSCa framework is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

The MuSCa framework is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with the MuSCa framework.  If not, see <http://www.gnu.org/licenses/>.

Initial developer(s): Sam Rottenberg
Contributor(s):
-->
<mscharacterization:MSCharacterization xmi:version="2.0" xmlns:xmi="http://
    www.omg.org/XMI" xmlns:mscharacterization="http://www-public.it-sudparis.
    eu/~rottenbe/mscharacterization/3.0">
  <containsMeasures unity="OSMNominatimLocationRank" valueType="java.lang.
      Integer"/>
  <containsMeasures unity="Bytes" valueType="java.lang.Double"
      toMeasureConverters="//@containsMeasureConverters.0"/>
  <containsMeasures unity="MegaBytes" valueType="java.lang.Long"
      fromMeasureConverters="//@containsMeasureConverters.0"/>
  <containsMeasures unity="KBPS" valueType="java.lang.Double"/>
  <containsMeasures unity="Millisecond" valueType="java.lang.Long"/>
  <containsMeasures unity="Meter" valueType="java.lang.Double"
      toMeasureConverters="//@containsMeasureConverters.1"/>
  <containsMeasures unity="KiloMeter" valueType="java.lang.Double"
      fromMeasureConverters="//@containsMeasureConverters.1"/>
  <containsMeasures unity="MIPS" valueType="java.lang.Double"/>
```

```
32  <containsMeasures unity="MembershipPolicyUnity" valueType="java.lang.String
        "/>
33  <containsMeasures unity="Number" valueType="java.lang.Long"/>
34  <containsMeasures unity="SmallestCommonLocationUnity" valueType="java.lang.
        String"/>
35  <containsViewpoints name="Device">
36    <viewpointContainsDimensions name="StorageCapacity"
          dimensionHasBasicProbe="//@containsBasicProbes.1">
37      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .1">
38        <scaleSetContainsScales name="Kilo" min="0" max="500000000"/>
39        <scaleSetContainsScales name="Giga" min="500000000" max
              ="1000000000000000"/>
40        <scaleSetContainsScales name="Peta" min="1000000000000000" max="
              Infinity"/>
41      </dimensionContainsScaleSets>
42    </viewpointContainsDimensions>
43    <viewpointContainsDimensions name="ProcessingPower">
44      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .7">
45        <scaleSetContainsScales name="Kilo"/>
46        <scaleSetContainsScales name="Giga"/>
47        <scaleSetContainsScales name="Peta"/>
48      </dimensionContainsScaleSets>
49    </viewpointContainsDimensions>
50    <projectionProperties>id</projectionProperties>
51  </containsViewpoints>
52  <containsViewpoints name="Network">
53    <viewpointContainsDimensions name="NetworkRange">
54      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .5">
55        <scaleSetContainsScales name="PAN"/>
56        <scaleSetContainsScales name="LAN"/>
57        <scaleSetContainsScales name="MAN"/>
58        <scaleSetContainsScales name="WAN"/>
59      </dimensionContainsScaleSets>
60    </viewpointContainsDimensions>
61    <viewpointContainsDimensions name="Bandwidth">
62      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .3">
63        <scaleSetContainsScales name="LowBandwidth"/>
64        <scaleSetContainsScales name="MediumBandwidth"/>
65        <scaleSetContainsScales name="HighBandwidth"/>
66      </dimensionContainsScaleSets>
67    </viewpointContainsDimensions>
```

```
68    <viewpointContainsDimensions name="Latency">
69      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .4">
70        <scaleSetContainsScales name="LowLatency"/>
71        <scaleSetContainsScales name="HighLatency"/>
72      </dimensionContainsScaleSets>
73    </viewpointContainsDimensions>
74  </containsViewpoints>
75  <containsViewpoints name="Data"/>
76  <containsViewpoints name="Geography">
77    <viewpointContainsDimensions name="Location" treeDimensionName="Location"
          getScaleMethodHasParameters="true" dimensionHasBasicProbe="//
          @containsBasicProbes.0">
78      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .0">
79        <scaleSetContainsScales name="Point" min="50" max="50"/>
80        <scaleSetContainsScales name="Building" min="28" max="49"/>
81        <scaleSetContainsScales name="Street" min="26" max="27"/>
82        <scaleSetContainsScales name="Neighborhood" min="22" max="25"/>
83        <scaleSetContainsScales name="Suburb" min="20" max="21"/>
84        <scaleSetContainsScales name="City" min="16" max="19"/>
85        <scaleSetContainsScales name="Region" min="10" max="15"/>
86        <scaleSetContainsScales name="Country" min="4" max="9"/>
87        <scaleSetContainsScales name="World" min="0" max="3"/>
88      </dimensionContainsScaleSets>
89    </viewpointContainsDimensions>
90    <viewpointContainsDimensions name="Distance" getScaleMethodHasParameters
          ="true" dimensionHasBasicProbe="//@containsBasicProbes.2">
91      <dimensionContainsScaleSets scaleSetHasInScaleRule="//
            @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
            .5">
92        <scaleSetContainsScales name="Under100m" min="0" max="100"/>
93        <scaleSetContainsScales name="Between_100m_And_5km" min="100" max
              ="5000"/>
94        <scaleSetContainsScales name="Between_5km_And_50km" min="5000" max
              ="50000"/>
95        <scaleSetContainsScales name="Between_50km_And_1000km" min="50000"
              max="1000000"/>
96        <scaleSetContainsScales name="Over_1000km" min="1000000" max="
              Infinity"/>
97      </dimensionContainsScaleSets>
98    </viewpointContainsDimensions>
99    <projectionProperties>id</projectionProperties>
100   <projectionProperties>lat</projectionProperties>
101   <projectionProperties>lon</projectionProperties>
102   <projectionProperties>lat_lon</projectionProperties>
103 </containsViewpoints>
```

```
104 <containsViewpoints name="User">
105   <viewpointContainsDimensions name="MembershipPolicy">
106     <dimensionContainsScaleSets scaleSetHasInScaleRule="//
          @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
          .8">
107       <scaleSetContainsScales name="Individual"/>
108       <scaleSetContainsScales name="Group"/>
109       <scaleSetContainsScales name="Community"/>
110       <scaleSetContainsScales name="All"/>
111     </dimensionContainsScaleSets>
112   </viewpointContainsDimensions>
113   <viewpointContainsDimensions name="NumberOfUsers">
114     <dimensionContainsScaleSets scaleSetHasInScaleRule="//
          @containsInScaleRules.0" scaleSetHasMeasure="//@containsMeasures
          .9">
115       <scaleSetContainsScales name="Group"/>
116       <scaleSetContainsScales name="Community"/>
117       <scaleSetContainsScales name="All"/>
118     </dimensionContainsScaleSets>
119   </viewpointContainsDimensions>
120 </containsViewpoints>
121 <containsInScaleRules name="InExactBounds" isInScaleCondition_VALUE_SCALE
      ="(VALUE.compareTo(SCALE.min) >= 0) &amp;amp;amp; (VALUE.compareTo(SCALE.
      max) &lt;= 0)"/>
122 <containsInScaleRules name="InApproximateBounds"/>
123 <containsInScaleRules name="InSemanticBounds"/>
124 <containsBasicProbes mavenArtifact="//@containsMavenArtifacts.1"
      fullyQualifiedClassName="osmgeoprobes.client.GeoClient" information="
      Location_Smallest_Common_Scale_Rank" treeInformation="
      Location_Scale_Path" basicProbeHasMeasure="//@containsMeasures.0"
      initialization_PROBE="PROBE.setId(this.id);&#xA;PROBE.setLat(this.lat)
      ;&#xA;PROBE.setLon(this.lon);&#xA;PROBE.registerToServer();">
125   <parameter parameterType="java.lang.String"/>
126   <constructorProperties>server_uri</constructorProperties>
127 </containsBasicProbes>
128 <containsBasicProbes mavenArtifact="//@containsMavenArtifacts.0"
      fullyQualifiedClassName="fr.enac.lii.basicprobes.HDProbe" information="
      HD_TOTALSPACE" basicProbeHasMeasure="//@containsMeasures.2"/>
129 <containsBasicProbes mavenArtifact="//@containsMavenArtifacts.1"
      fullyQualifiedClassName="osmgeoprobes.client.GeoDistanceProbe"
      information="MAX_DISTANCE" basicProbeHasMeasure="//@containsMeasures.6"
       initialization_PROBE="PROBE.setLat(this.lat);PROBE.setLon(this.lon);">
130   <parameter parameterType="java.lang.String" valueFromProjectionField="
        lat_lon"/>
131 </containsBasicProbes>
132 <containsMeasureConverters fromMeasure="//@containsMeasures.2" toMeasure
      ="//@containsMeasures.1" conversionFunction_VALUE="VALUE * 1024 *
      1024"/>
133 <containsMeasureConverters fromMeasure="//@containsMeasures.6" toMeasure
```

```
      ="//@containsMeasures.5" conversionFunction_VALUE="VALUE * 1000"/>
134   <containsMavenArtifacts mavenGroupId="fr.enac.lii.basicprobes"
      mavenArtifactId="fr.enac.lii.basicprobes" mavenVersion="0.0.2-SNAPSHOT
      "/>
135   <containsMavenArtifacts mavenGroupId="osmgeoprobes" mavenArtifactId="client
      " mavenVersion="0.0.1-SNAPSHOT"/>
136 </mscharacterization:MSCharacterization>
```

Annexe B

Templates Acceleo pour la génération des sondes multiéchelles

---

Cette annexe présente une sélection des cinq *templates* Acceleo les plus représentatifs, parmi les *templates* que nous avons définis pour la génération de sondes multiéchelles. Notons que ces *templates* font appel à des fonctions (appelées *queries*) ou à d'autres *templates* utilitaires que nous avons définis pour alléger les *templates* principaux, et qui ne sont pas intégrés dans cette annexe.

Le premier *template* est le *template* principal. C'est le point d'entrée du processus de génération des sondes multiéchelles.

Le deuxième *template* permet de générer un fichier `pom.xml` (fichier de configuration de module Maven [1]) pour les sondes multiéchelles générées. Il permet d'illustrer l'utilisation de la classe `MavenArtifact` du métamodèle MuSCa_Probes (cf. section 5.3.2) pour intégrer les modules Maven des sondes de bas niveau au processus de compilation et de construction des sondes multiéchelles générées.

Le troisième *template* est utilisé pour générer, pour chaque point de vue d'une caractérisation MuSCa, une classe Java qui représente la sonde multiéchelle pour ce point de vue. Ces classes constituent les points d'accès aux sondes multiéchelles pour une application cliente.

Le quatrième *template* génère, pour chaque point de vue, une interface Java représentant une entité mesurable pour ce point de vue, c'est-à-dire une projection d'entité du système sur ce point de vue.

1. http://maven.apache.org/

Enfin, le cinquième *template* (dont un extrait est présenté figure 5.14, section 5.4.2) permet de générer, pour chaque ensemble d'échelles d'une caractérisation MuSCa, une énumération Java qui représente cet ensemble d'échelles. Les éléments de ces énumérations, qui représentent les échelles, constituent les valeurs retournées aux applications clientes lors de l'appel d'une sonde multiéchelle.

```
1  [comment encoding = UTF-8 /]
2  [comment]
3          Copyright (c) 2011, 2014 Institut Mines-Télécom / Télécom SudParis.
4  [/comment]
5  [comment]
6  /**
7   * This file is part of the INCOME platform.
8   *
9   * The INCOME platform is free software: you can redistribute it and/or
        modify
10  * it under the terms of the GNU Lesser General Public License as published
        by
11  * the Free Software Foundation, either version 3 of the License, or
12  * (at your option) any later version.
13  *
14  * The INCOME platform is distributed in the hope that it will be useful,
15  * but WITHOUT ANY WARRANTY; without even the implied warranty of
16  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17  * GNU Lesser General Public License for more details.
18  *
19  * You should have received a copy of the GNU Lesser General Public License
20  * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
        />.
21  *
22  * Initial developer(s): Sam Rottenberg
23  * Contributor(s):
24  */
25 [/comment]
26 [module genMSCharacterization('http://www-public.it-sudparis.eu/~rottenbe/
        mscharacterization/3.0')]

28 [import MSCharacterizationGenerator::files::genPOM /]
29 [import MSCharacterizationGenerator::files::genMeasureValue /]
30 [import MSCharacterizationGenerator::files::genMeasureConverter /]
31 [import MSCharacterizationGenerator::files::
        genBasicProbeConstructorProperties /]
32 [import MSCharacterizationGenerator::files::genViewpointProperties /]
33 [import MSCharacterizationGenerator::files::genIViewpointMeasurable /]
34 [import MSCharacterizationGenerator::files::
        genViewpointMeasurableElementProjection /]
35 [import MSCharacterizationGenerator::files::genViewpointProbe /]
```

```
36 [import MSCharacterizationGenerator::files::genScaleEnum /]
37 [import MSCharacterizationGenerator::files::genScaleTree /]
38 [import MSCharacterizationGenerator::services::utilServices /]

40 [template public genMSCharacterization(aMSCharacterization :
       MSCharacterization)]
41 [comment @main/]
42 [genPOM(aMSCharacterization)/]
43 [genMeasureConverter(aMSCharacterization)/]

45 [for (measure : Measure | getMeasures(aMSCharacterization))]
46         [genMeasureValue(measure)/]
47 [/for]

49 [for (probe : BasicProbe | getBasicProbes(aMSCharacterization))]
50         [genBasicProbeConstructorProperties(probe)/]
51 [/for]

53 [for (aViewpoint : Viewpoint | aMSCharacterization.containsViewpoints)]
54         [genViewpointProperties(aViewpoint)/]
55         [genIViewpointMeasurable(aViewpoint)/]
56         [genViewpointMeasurableElementProjection(aViewpoint)/]
57         [genViewpointProbe(aViewpoint)/]
58         [for (dimension : Dimension | aViewpoint.viewpointContainsDimensions)
               ]
59                 [for (scaleSet : ScaleSet | dimension.
                       dimensionContainsScaleSets)]
60                         [genScaleEnum(scaleSet)/]
61                         [if (not scaleSet.scaleSetIsContainedInDimension.
                               treeDimensionName.oclIsUndefined())]
62                                 [genScaleTree(scaleSet)/]
63                         [/if]
64                 [/for]
65         [/for]
66 [/for]

68 [/template]
```

```
1  [comment encoding = UTF-8 /]
2  [comment]
3          Copyright (c) 2011, 2014 Institut Mines-Télécom / Télécom SudParis.
4  [/comment]
5  [comment]
6  /**
7   * This file is part of the INCOME platform.
8   *
9   * The INCOME platform is free software: you can redistribute it and/or
        modify
10  * it under the terms of the GNU Lesser General Public License as published
        by
```

```
11  * the Free Software Foundation, either version 3 of the License, or
12  * (at your option) any later version.
13  *
14  * The INCOME platform is distributed in the hope that it will be useful,
15  * but WITHOUT ANY WARRANTY; without even the implied warranty of
16  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17  * GNU Lesser General Public License for more details.
18  *
19  * You should have received a copy of the GNU Lesser General Public License
20  * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
        />.
21  *
22  * Initial developer(s): Sam Rottenberg
23  * Contributor(s):
24  */
25 [/comment]
26 [module genPOM('http://www-public.it-sudparis.eu/~rottenbe/mscharacterization
        /3.0')/]

28 [import MSCharacterizationGenerator::services::utilServices /]

30 [template public genPOM(aMSCharacterization : MSCharacterization)]
31 [file ('../pom.xml', false, 'UTF-8')]
32 <!--
33 This file is part of the INCOME platform.

35 The INCOME platform is free software: you can redistribute it and/or modify
36 it under the terms of the GNU Lesser General Public License as published by
37 the Free Software Foundation, either version 3 of the License, or
38 (at your option) any later version.

40 The INCOME platform is distributed in the hope that it will be useful,
41 but WITHOUT ANY WARRANTY; without even the implied warranty of
42 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
43 GNU Lesser General Public License for more details.

45 You should have received a copy of the GNU Lesser General Public License
46 along with the INCOME platform.  If not, see <http://www.gnu.org/licenses/>.

48 Initial developer(s): Sam Rottenberg
49 Contributor(s):
50 -->
51 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
       org/2001/XMLSchema-instance"
52          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
                apache.org/xsd/maven-4.0.0.xsd">
53          <modelVersion>4.0.0</modelVersion>
54          <groupId>eu.it-sudparis</groupId>
55          <artifactId>GeneratedScaleAwarenessProbe</artifactId>
```

```
56          <version>0.0.1-SNAPSHOT</version>
57          <build>
58                  <sourceDirectory>src</sourceDirectory>
59                  <resources>
60                          <resource>
61                                  <directory>src</directory>
62                                  <excludes>
63                                          <exclude>**/*.java</exclude>
64                                  </excludes>
65                          </resource>
66                  </resources>
67                  <pluginManagement>
68                          <plugins>
69                                  <plugin>
70                                          <groupId>org.apache.maven.plugins</
                                                groupId>
71                                          <artifactId>maven-compiler-plugin</
                                                artifactId>
72                                          <version>3.1</version>
73                                          <configuration>
74                                                  <source>1.7</source>
75                                                  <target>1.7</target>
76                                          </configuration>
77                                  </plugin>
78                          </plugins>
79                  </pluginManagement>
80          </build>
81          <dependencies>
82                  <dependency>
83                          <groupId>fr.irit.devext.probesframework</groupId>
84                          <artifactId>fr.irit.devext.probesframework</
                                artifactId>
85                          <version>0.0.2-SNAPSHOT</version>
86                  </dependency>
87                  [for (aMavenArtifact : MavenArtifact | getMavenArtifacts(
                        aMSCharacterization))]
88                  <dependency>
89                          <groupId>[aMavenArtifact.mavenGroupId/]</groupId>
90                          <artifactId>[aMavenArtifact.mavenArtifactId/]</
                                artifactId>
91                          <version>[aMavenArtifact.mavenVersion/]</version>
92                  </dependency>
93                  [/for]
94          </dependencies>
95  </project>
96  [/file]
97  [/template]
```

```
1  [comment encoding = UTF-8 /]
2  [comment]
```

```
3         Copyright (c) 2011, 2014 Institut Mines-Télécom / Télécom SudParis.
4 [/comment]
5 [comment]
6 /**
7  * This file is part of the INCOME platform.
8  *
9  * The INCOME platform is free software: you can redistribute it and/or
       modify
10 * it under the terms of the GNU Lesser General Public License as published
       by
11 * the Free Software Foundation, either version 3 of the License, or
12 * (at your option) any later version.
13 *
14 * The INCOME platform is distributed in the hope that it will be useful,
15 * but WITHOUT ANY WARRANTY; without even the implied warranty of
16 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17 * GNU Lesser General Public License for more details.
18 *
19 * You should have received a copy of the GNU Lesser General Public License
20 * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
       />.
21 *
22 * Initial developer(s): Sam Rottenberg
23 * Contributor(s):
24 */
25 [/comment]
26 [module genViewpointProbe('http://www-public.it-sudparis.eu/~rottenbe/
       mscharacterization/3.0')]

28 [import MSCharacterizationGenerator::util::utilTemplatesAndQueries /]
29 [import MSCharacterizationGenerator::services::utilServices /]

31 [template public genViewpointProbe(aViewpoint : Viewpoint)]

33 [file (queryViewpointProbeFileName(aViewpoint), false, 'UTF-8')]
34 /**
35  * This file is part of the INCOME platform.
36  *
37  * The INCOME platform is free software: you can redistribute it and/or
       modify
38 * it under the terms of the GNU Lesser General Public License as published
       by
39 * the Free Software Foundation, either version 3 of the License, or
40 * (at your option) any later version.
41 *
42 * The INCOME platform is distributed in the hope that it will be useful,
43 * but WITHOUT ANY WARRANTY; without even the implied warranty of
44 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
45 * GNU Lesser General Public License for more details.
```

```
46  *
47  * You should have received a copy of the GNU Lesser General Public License
48  * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
        />.
49  *
50  * Initial developer(s): Sam Rottenberg
51  * Contributor(s):
52  */
53 package msawareness.viewpoints.[aViewpoint.name.toLowerCase()/];

55 import fr.irit.devext.probesframework.AbstractProbe;
56 import fr.irit.devext.probesframework.IncorrectTypeException;

58 public class [queryViewpointProbeName(aViewpoint)/] extends AbstractProbe {

60         public enum Information {
61         [for (aScaleSet : ScaleSet | getViewpointContainsScaleSets(aViewpoint
               )) separator (',\n')]
62                 [queryProbeInformationName(aViewpoint, aScaleSet)/]
63                 [if (not aScaleSet.scaleSetIsContainedInDimension.
                       treeDimensionName.oclIsUndefined())]
64                 ,
65                 [queryProbeTreeInformationName(aViewpoint, aScaleSet)/]
66                 [/if]

68         [/for]
69         }

71         private [queryIViewpointMeasurableName(aViewpoint)/] element;

73         public [queryViewpointProbeName(aViewpoint)/]() {
74                 super(1000 * 60 * 60);
75                 // TODO A garder ?

77                 element = new [queryViewpointMeasurableElementProjectionName(
                       aViewpoint)/]();

79                 element.initializeBasicProbes();
80         }

82         @Override
83         protected void addInformation() {
84                 try {
85                 [for (aScaleSet : ScaleSet | getViewpointContainsScaleSets(
                       aViewpoint))]
86                         addInformation(Information.[queryProbeInformationName
                               (aViewpoint, aScaleSet)/].ordinal(),
87                                         "[queryGetScaleMethodName(aScaleSet)
                                               /]"[templateInformationArgs(
```

```
                                                aScaleSet)/]);
88                          [if (not aScaleSet.scaleSetIsContainedInDimension.
                                treeDimensionName.oclIsUndefined())]
89                          addInformation(Information.[
                                queryProbeTreeInformationName(aViewpoint,
                                aScaleSet)/].ordinal(),
90                                          "[queryGetScaleTreeMethodName(
                                               aScaleSet)/]");
91                          [/if]
92                  [/for]
93                  } catch (NoSuchMethodException e) {
94                          e.printStackTrace();
95                  } catch (IncorrectTypeException e) {
96                          e.printStackTrace();
97                  }
98          }

100         @Override
101         public boolean exists() {
102                 return true;
103         }

105         @Override
106         public boolean isActive() {
107                 return true;
108         }

110         [for (aScaleSet : ScaleSet | getViewpointContainsScaleSets(aViewpoint
                ))]
111         public [queryScaleEnumName(aScaleSet)/] [queryGetScaleMethodName(
                aScaleSet)/]([templateGetScaleMethodParametersWithType(aScaleSet)
                /]) {
112                 return element.[queryGetScaleMethodName(aScaleSet)/]([
                        templateGetScaleMethodParameters(aScaleSet)/]);
113         }

115         [if (not aScaleSet.scaleSetIsContainedInDimension.treeDimensionName.
                oclIsUndefined())]
116         public [queryScaleTreeName(aScaleSet)/] [queryGetScaleTreeMethodName(
                aScaleSet)/]() {
117                 return element.[queryGetScaleTreeMethodName(aScaleSet)/]();
118         }
119         [/if]
120         [/for]

122         public [queryIViewpointMeasurableName(aViewpoint)/] getElement() {
123                 return this.element;
124         }
125 }
```

```
128 [/file]

130 [/template]
```

```
1  [comment encoding = UTF-8 /]
2  [comment]
3          Copyright (c) 2011, 2014 Institut Mines-Télécom / Télécom SudParis.
4  [/comment]
5  [comment]
6  /**
7   * This file is part of the INCOME platform.
8   *
9   * The INCOME platform is free software: you can redistribute it and/or
        modify
10  * it under the terms of the GNU Lesser General Public License as published
        by
11  * the Free Software Foundation, either version 3 of the License, or
12  * (at your option) any later version.
13  *
14  * The INCOME platform is distributed in the hope that it will be useful,
15  * but WITHOUT ANY WARRANTY; without even the implied warranty of
16  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17  * GNU Lesser General Public License for more details.
18  *
19  * You should have received a copy of the GNU Lesser General Public License
20  * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
        />.
21  *
22  * Initial developer(s): Sam Rottenberg
23  * Contributor(s):
24  */
25 [/comment]
26 [module genIViewpointMeasurable('http://www-public.it-sudparis.eu/~rottenbe/
        mscharacterization/3.0')]

28 [import MSCharacterizationGenerator::util::utilTemplatesAndQueries /]
29 [import MSCharacterizationGenerator::services::utilServices /]

31 [template public genIViewpointMeasurable(aViewpoint : Viewpoint)]

33 [file (queryIViewpointMeasurableFileName(aViewpoint), false, 'UTF-8')]
34 /**
35  * This file is part of the INCOME platform.
36  *
37  * The INCOME platform is free software: you can redistribute it and/or
        modify
38  * it under the terms of the GNU Lesser General Public License as published
        by
```

```
39  * the Free Software Foundation, either version 3 of the License, or
40  * (at your option) any later version.
41  *
42  * The INCOME platform is distributed in the hope that it will be useful,
43  * but WITHOUT ANY WARRANTY; without even the implied warranty of
44  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
45  * GNU Lesser General Public License for more details.
46  *
47  * You should have received a copy of the GNU Lesser General Public License
48  * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
        />.
49  *
50  * Initial developer(s): Sam Rottenberg
51  * Contributor(s):
52  */
53 package msawareness.viewpoints.[aViewpoint.name.toLowerCase()/];

55 import msawareness.measures.*;


58 public interface [queryIViewpointMeasurableName(aViewpoint)/] {

60 [for (dimension : Dimension | aViewpoint.viewpointContainsDimensions)]
61         [for (scaleSet : ScaleSet | dimension.dimensionContainsScaleSets)]
62         public [queryMeasureValueName(scaleSet.scaleSetHasMeasure)/]
               getValue_[queryDimensionInMeasure()/]([
               templateGetScaleMethodParametersWithType(scaleSet)/]);
63         public [queryScaleEnumName(scaleSet)/] [queryGetScaleMethodName(
               scaleSet)/]([templateGetScaleMethodParametersWithType(scaleSet)
               /]);

65         [if (not scaleSet.scaleSetIsContainedInDimension.treeDimensionName.
               oclIsUndefined())]
66         public [queryScaleTreeName(scaleSet)/] [queryGetScaleTreeMethodName(
               scaleSet)/]();
67         [/if]
68         [/for]
69 [/for]

71 [comment][for (param : BasicProbeInformationParameter |
       getBasicProbeInformationParametersFromViewpoint(aViewpoint)) before ('\t
       // Move to abstract class ?\n')]
72         [if (not param.oclIsUndefined())]
73         public [getBasicProbeParameterType(param)/] get[param.
               valueFromProjectionField.toUpperFirst()/]();
74         [/if]
75 [/for][/comment]

77 [for (prop : String | aViewpoint.projectionProperties)]
```

```
78          public String get[prop.toUpperFirst()/]();
79  [/for]

81          public void initializeBasicProbes();

83  }
84  [/file]
85  [/template]
```

```
1   [comment encoding = UTF-8 /]
2   [comment]
3           Copyright (c) 2011, 2014 Institut Mines-Télécom / Télécom SudParis.
4   [/comment]
5   [comment]
6   /**
7    * This file is part of the INCOME platform.
8    *
9    * The INCOME platform is free software: you can redistribute it and/or
         modify
10   * it under the terms of the GNU Lesser General Public License as published
         by
11   * the Free Software Foundation, either version 3 of the License, or
12   * (at your option) any later version.
13   *
14   * The INCOME platform is distributed in the hope that it will be useful,
15   * but WITHOUT ANY WARRANTY; without even the implied warranty of
16   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17   * GNU Lesser General Public License for more details.
18   *
19   * You should have received a copy of the GNU Lesser General Public License
20   * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
         />.
21   *
22   * Initial developer(s): Sam Rottenberg
23   * Contributor(s):
24   */
25  [/comment]
26  [module genScaleEnum('http://www-public.it-sudparis.eu/~rottenbe/
        mscharacterization/3.0')]

28  [import MSCharacterizationGenerator::util::utilTemplatesAndQueries /]

30  [template public genScaleEnum(aScaleSet : ScaleSet)]
31  [file (queryScaleEnumFileName(aScaleSet), false, 'UTF-8')]
32  /**
33   * This file is part of the INCOME platform.
34   *
35   * The INCOME platform is free software: you can redistribute it and/or
         modify
36   * it under the terms of the GNU Lesser General Public License as published
```

```
     by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * The INCOME platform is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with the INCOME platform.  If not, see <http://www.gnu.org/licenses
     />.
 *
 * Initial developer(s): Sam Rottenberg
 * Contributor(s):
 */
package msawareness.viewpoints.[aScaleSet.scaleSetIsContainedInDimension.
    dimensionIsContainedInViewpoint.name.toLowerCase()/];

import msawareness.measures.*;

public enum [queryScaleEnumName(aScaleSet)/] {
[for (scale : Scale | aScaleSet.scaleSetContainsScales) separator(',\n')]
        [scale.name.toUpperCase()/]("[scale.min/]", "[scale.max/]")
[/for];

        private final [queryMeasureValueName(aScaleSet.scaleSetHasMeasure)/]
            min;
        private final [queryMeasureValueName(aScaleSet.scaleSetHasMeasure)/]
            max;

        [queryScaleEnumName(aScaleSet)/](String min, String max) {
                this.min = new [queryMeasureValueName(aScaleSet.
                    scaleSetHasMeasure)/](min);
                this.max = new [queryMeasureValueName(aScaleSet.
                    scaleSetHasMeasure)/](max);
        }

        public [queryMeasureValueName(aScaleSet.scaleSetHasMeasure)/] getMin
            () {
                return this.min;
        }

        public [queryMeasureValueName(aScaleSet.scaleSetHasMeasure)/] getMax
            () {
                return this.max;
        }
```

```
77          public static [queryScaleEnumName(aScaleSet)/]
                getScaleFromMeasureValue([queryMeasureValueName(aScaleSet.
                scaleSetHasMeasure)/] value) {
78              for ([queryScaleEnumName(aScaleSet)/] scale : [
                    queryScaleEnumName(aScaleSet)/].values()) {
79                  if (scale.isInScale(value)) {
80                          return scale;
81                  }
82              }
83              return null;
84          }

86          private boolean isInScale([queryMeasureValueName(aScaleSet.
                scaleSetHasMeasure)/] value) {
87              return [aScaleSet.scaleSetHasInScaleRule.
                    isInScaleCondition_VALUE_SCALE.replaceAll('VALUE', 'value
                    ').replaceAll('SCALE', 'this')/];
88          }

90  }
91  [/file]
92  [/template]
```

Annexe C

Production de rapports multiéchelles

```
1  /**
2  This file is part of the INCOME platform.
3
4  The INCOME platform is free software: you can redistribute it and/or modify
5  it under the terms of the GNU Lesser General Public License as published by
6  the Free Software Foundation, either version 3 of the License, or
7  (at your option) any later version.
8
9  The INCOME platform is distributed in the hope that it will be useful,
10 but WITHOUT ANY WARRANTY; without even the implied warranty of
11 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12 GNU Lesser General Public License for more details.
13
14 You should have received a copy of the GNU Lesser General Public License
15 along with the INCOME platform.  If not, see <http://www.gnu.org/licenses/>.
16
17 Initial developer(s): Sam Rottenberg
18 Contributor(s):
19  */
20
21 package mucontext.datamodel.multiscalability;
22
23 import java.util.TreeMap;
24
25 import javax.xml.bind.annotation.XmlRootElement;
26
27 import msawareness.viewpoints.device.DeviceProbe;
28 import msawareness.viewpoints.device.StorageCapacity_In_Bytes_Scale;
29 import msawareness.viewpoints.geography.GeographyProbe;
30 import msawareness.viewpoints.geography.
       Location_In_OSMNominatimLocationRank_ScaleTree;
31
32 @XmlRootElement
33 public class MuSCaMultiscalabilityReport extends MultiscalabilityReport {
34         public GeographyViewpointReport geographyViewpointReport;
```

```
35      public DeviceViewpointReport deviceViewpointReport;

37          MuSCaMultiscalabilityReport(final String id) {
38                  super(id);
39          }

41          MuSCaMultiscalabilityReport() {
42                  this("");
43          }

45          MuSCaMultiscalabilityReport addGeographyViewpointReport(
                GeographyViewpointReport geographyViewpointReport) {
46                  this.geographyViewpointReport = geographyViewpointReport;
47                  return this;
48          }

50          MuSCaMultiscalabilityReport addDeviceViewpointReport(
                DeviceViewpointReport deviceViewpointReport) {
51                  this.deviceViewpointReport = deviceViewpointReport;
52                  return this;
53          }

55          @Override
56          public int hashCode() {
57                  final int prime = 31;
58                  int result = 1;
59                  result = prime * result + ((id == null) ? 0 : id.hashCode());
60                  return result;
61          }

63          @Override
64          public boolean equals(Object obj) {
65                  if (this == obj) {
66                          return true;
67                  }
68                  if (obj == null) {
69                          return false;
70                  }
71                  if (!(obj instanceof MuSCaMultiscalabilityReport)) {
72                          return false;
73                  }
74                  MuSCaMultiscalabilityReport other = (
                        MuSCaMultiscalabilityReport) obj;
75                  if (id == null) {
76                          if (other.id != null) {
77                                  return false;
78                          }
79                  } else if (!id.equals(other.id)) {
80                          return false;
```

```
81                  }
82                  return true;
83          }

85          @Override
86          public String toString() {
87                  return "MultiscalabilityReport␣[id=" + id + "]";
88          }


91  }

93  class GeographyViewpointReport {
94      public GeographyProjection geographyProjection;
95      public Location_In_OSMNominatimLocationRank_ScaleTree
            location_In_OSMNominatimLocationRank_ScaleTree;

97      public GeographyViewpointReport(GeographyProbe probe) {
98                  this.geographyProjection = new GeographyProjection(probe);

100                 if(probe != null) {
101                         this.location_In_OSMNominatimLocationRank_ScaleTree =
                                (Location_In_OSMNominatimLocationRank_ScaleTree)
                                probe.getInformation(GeographyProbe.Information.
                               GEOGRAPHY_LOCATION_IN_OSMNOMINATIMLOCATIONRANK_SCALE_TREE
                               .ordinal());
102                 }
103         }

105     public GeographyViewpointReport() {
106         this(null);
107     }

109 }

111 class GeographyProjection {
112         public String id;
113     public String latitude;
114     public String longitude;

116         public GeographyProjection(GeographyProbe probe) {
117                 if (probe != null && probe.getElement() != null) {
118                         this.id = probe.getElement().getId();
119                         this.latitude = probe.getElement().getLat();
120                         this.longitude = probe.getElement().getLon();
121                 }
122         }

124         public GeographyProjection() {
```

```
125                 this(null);
126         }
127 }

129 class DeviceViewpointReport {
130       public DeviceProjection deviceProjection;
131       public StorageCapacity_In_Bytes_Scale storageCapacity_In_Bytes_Scale;

133       public DeviceViewpointReport(DeviceProbe probe) {
134           this.deviceProjection = new DeviceProjection(probe);
135           if (probe != null) {
136                   this.storageCapacity_In_Bytes_Scale = (
                        StorageCapacity_In_Bytes_Scale) probe.getInformation(
                        DeviceProbe.Information.
                        DEVICE_STORAGECAPACITY_IN_BYTES_SCALE.ordinal());
137           }
138       }

140       public DeviceViewpointReport() {
141           this(null);
142       }
143 }

145 class DeviceProjection {
146         public DeviceProjection(DeviceProbe probe) {}

148         public DeviceProjection() {
149                 this(null);
150         }
151 }
```

```
1  /**
2  This file is part of the INCOME platform.

4  The INCOME platform is free software: you can redistribute it and/or modify
5  it under the terms of the GNU Lesser General Public License as published by
6  the Free Software Foundation, either version 3 of the License, or
7  (at your option) any later version.

9  The INCOME platform is distributed in the hope that it will be useful,
10 but WITHOUT ANY WARRANTY; without even the implied warranty of
11 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12 GNU Lesser General Public License for more details.

14 You should have received a copy of the GNU Lesser General Public License
15 along with the INCOME platform.  If not, see <http://www.gnu.org/licenses/>.

17 Initial developer(s): Denis Conan
18 Contributor(s):
19  */
```

```
21 package mucontext.datamodel.multiscalability;

23 import java.io.File;
24 import java.io.StringReader;
25 import java.io.StringWriter;

27 import javax.xml.bind.JAXBContext;
28 import javax.xml.bind.JAXBException;
29 import javax.xml.bind.Marshaller;
30 import javax.xml.bind.Unmarshaller;

32 import msawareness.viewpoints.device.DeviceProbe;
33 import msawareness.viewpoints.geography.GeographyProbe;
34 import msawareness.viewpoints.geography.
       SmallestCommonLocation_In_OSMNominatimLocationRank_Scale;

36 public class MuSCaMultiscalabilityDataModelFacade {
37         /**
38          * the name of the facade.
39          */
40         private String name;
41         /**
42          * the set of default classes for the JAXB classes.
43          */
44         private static Class<?>[] defaultClasses = new Class<?>[] {
45                         EventDistributionRange.class,
                                MuSCaMultiscalabilityReport.class,
                                SmallestCommonLocation_In_OSMNominatimLocationRank_Scale
                                .class };
46         /**
47          * JAXB marshaller for multiscalability reports.
48          */
49         private Marshaller muSCaMultiscalabilityReportMarshaller;

51         /**
52          * JAXB unmarshaller for multiscalability reports.
53          */
54         private Unmarshaller muSCaMultiscalabilityReportUnmarshaller;

56         public MuSCaMultiscalabilityDataModelFacade(final String name,
57                         final Class<?>[] otherClasses) {
58                 this.name = name;
59                 try {
60                         Class<?>[] allClasses = null;
61                         if (otherClasses == null) {
62                                 allClasses = defaultClasses;
63                         } else {
64                                 allClasses = new Class<?>[defaultClasses.
```

```
                                    length
65                                              + otherClasses.length];
66                          int i;
67                          for (i = 0; i < defaultClasses.length; i++) {
68                                  allClasses[i] = defaultClasses[i];
69                          }
70                          int j;
71                          for (j = 0; j < otherClasses.length; j++) {
72                                  allClasses[i + j] = otherClasses[j];
73                          }
74                  }
75                  JAXBContext jaxbContext = JAXBContext.newInstance(
                        allClasses);
76                  jaxbContext = JAXBContext.newInstance(allClasses);
77                  muSCaMultiscalabilityReportMarshaller = jaxbContext.
                        createMarshaller();
78                  muSCaMultiscalabilityReportMarshaller.setProperty(
79                                  Marshaller.JAXB_FORMATTED_OUTPUT,
                                        true);
80                  muSCaMultiscalabilityReportMarshaller.setProperty(
81                                  Marshaller.JAXB_ENCODING, "UTF-8");
82                  muSCaMultiscalabilityReportUnmarshaller = jaxbContext
83                                  .createUnmarshaller();
84          } catch (JAXBException e) {
85                  System.err.format("JAXBException:␣%s%n", e);
86          }
87  }

89  public MuSCaMultiscalabilityDataModelFacade(final String name) {
90          this(name, null);
91  }

93  public String getName() {
94          return name;
95  }

97  public MuSCaMultiscalabilityReport createMuSCaMultiscalabilityReport(
        final String id) {

99          GeographyViewpointReport geographyViewpointReport = new
                GeographyViewpointReport(new GeographyProbe());
100         DeviceViewpointReport deviceViewpointReport = new
                DeviceViewpointReport(new DeviceProbe());

102         MuSCaMultiscalabilityReport report = new
                MuSCaMultiscalabilityReport(id);
103         report.addGeographyViewpointReport(geographyViewpointReport);
104         report.addDeviceViewpointReport(deviceViewpointReport);
```

```
106                 return report;
107         }

109         public String serialiseToXML(final MuSCaMultiscalabilityReport report
                ) {
110                 try {
111                         java.io.StringWriter sw = new StringWriter();
112                         muSCaMultiscalabilityReportMarshaller.marshal(report,
                                sw);
113                         return sw.toString();
114                 } catch (JAXBException e) {
115                         System.err.format("JAXBException:␣%s%n", e);
116                 }
117                 return null;
118         }

120         public MuSCaMultiscalabilityReport unserialiseFromXML(final String
                serialised) {
121                 MuSCaMultiscalabilityReport result = null;
122                 try {
123                         result = (MuSCaMultiscalabilityReport)
                                muSCaMultiscalabilityReportUnmarshaller
124                                         .unmarshal(new StringReader(
                                                serialised));
125                 } catch (JAXBException e) {
126                         System.err.format("JAXBException:␣%s%n", e);
127                 }
128                 return result;
129         }

131         public final boolean systemOutExport(final
                MuSCaMultiscalabilityReport report) {
132                 boolean retValue = false;
133                 try {
134                         muSCaMultiscalabilityReportMarshaller.marshal(report,
                                System.out);
135                         retValue = true;
136                 } catch (JAXBException e) {
137                         System.err.format("JAXBException:␣%s%n", e);
138                 }
139                 return retValue;
140         }

142         public final boolean fileExport(final MuSCaMultiscalabilityReport
                report,
143                         final String filePath) {
144                 Boolean retValue = false;
145                 try {
146                         File file = new File(filePath);
```

```
147                         muSCaMultiscalabilityReportMarshaller.marshal(report,
                                file);
148                         retValue = true;
149                 } catch (JAXBException e) {
150                         System.err.format("JAXBException:␣%s%n", e);
151                 }
152                 return retValue;
153         }

155         public MuSCaMultiscalabilityReport fileImport(final String filePath)
                {
156                 MuSCaMultiscalabilityReport result = null;
157                 try {
158                         File file = new File(filePath);
159                         result = (MuSCaMultiscalabilityReport)
                                muSCaMultiscalabilityReportUnmarshaller
160                                         .unmarshal(file);
161                 } catch (JAXBException e) {
162                         System.err.format("JAXBException:␣%s%n", e);
163                 }
164                 return result;
165         }
166 }
```

Annexe D

Publications

Les résultats directs des travaux de cette thèse ont donné lieu aux publications suivantes :

- *Vers une définition des systèmes répartis multi-échelle* [Rottenberg 2012], présentée à Ubimob 2012 (conférence francophone à comité de lecture) ;
- *MuScA : A Multiscale Distributed Systems Scale-Awareness Framework* [Rottenberg 2014b], présentée à CAL 2014 (conférence francophone à comité de lecture) ;
- *MuSCa : A Multiscale Characterization Framework for Complex Distributed Systems* [Rottenberg 2014a], présentée à FedCSIS 2014 (conférence internationale à comité de lecture) ;
- *MuScADeL : A Deployment DSL based on a Multiscale Characterization Framework* [Boujbel 2014], présentée à COMPSAC Workshops 2014 (atelier international à comité de lecture).

De plus, l'intégration de ces travaux dans le projet INCOME a donné lieu à d'autres publications :

- *Gestion de contexte multi-échelle pour l'Internet des objets* [Arcangeli 2014b], présentée à Ubimob 2014 (conférence francophone à comité de lecture) ;
- *Projet INCOME : INfrastructure de gestion de COntexte Multi-Echelle pour l'Internet des Objets* [Arcangeli 2014a], présentée à CAL 2014 (conférence francophone à comité de lecture).

Bibliographie

[Arcangeli 2012] Jean-Paul Arcangeli, Amel Bouzeghoub, Valérie Camps, Marie-Françoise Canut, Sophie Chabridon, Denis Conan, Thierry Desprats, Romain Laborde, Emmanuel Lavinal, Sébastien Leriche, Hervé Maurel, André Péninou, Chantal Taconet and Pascale Zaraté. *INCOME – Multi-scale Context Management for the Internet of Things*. In Fabio Paternò, Boris Ruyter, Panos Markopoulos, Carmen Santoro, Evert Loenen and Kris Luyten, editeurs, Ambient Intelligence, volume 7683 of *Lecture Notes in Computer Science*, pages 338–347. Springer Berlin Heidelberg, 2012. (Cité en pages 15, 17 et 48.)

[Arcangeli 2013a] J.-P. Arcangeli, R. Boujbel, A. Bouzeghoub, V. Camps, S. Chabridon, D. Conan, T. Desprats, V. Guivarch, R. Laborde, E. Lavinal, S. Leriche, A. Oglaza, A. Péninou, S. Rottenberg, C. Taconet and P. Zaraté. *Architecture of Multiscalable Context Management — Version 1 — INCOME Livrable 3.1.a.1.* ANR INFRA INCOME Multi-Scale Context Management for the Internet of Things deliverable Task 3.1, Décembre 2013. (Cité en pages xv, 86, 87, 91, 92, 93, 94 et 95.)

[Arcangeli 2013b] J.-P. Arcangeli, R. Boujbel, A. Bouzeghoub, V. Camps, S. Chabridon, D. Conan, T. Desprats, V. Guivarch, R. Laborde, E. Lavinal, S. Leriche, A. Oglaza, A. Péninou, S. Rottenberg, C. Taconet and P. Zaraté. *Scénarios et Exigences — INCOME Livrable 2.2.* ANR INFRA INCOME Multi-Scale Context Management for the Internet of Things deliverable Task 2.2, Septembre 2013. (Cité en pages xv, 80, 81, 82, 84 et 85.)

[Arcangeli 2014a] Jean-Paul Arcangeli, Valérie Camps, Thierry Desprats, Romain Laborde, Emmanuel Lavinal, André Peninou, Pascale Zaraté, Raja Boujbel, Amel Bouzeghoub, Sophie Chabridon, Denis Conan, Chantal Taconet, Léon Lim, Samer Machara Marquez, Clément Mignard, Sam Rottenberg and Sébastien Leriche. *Projet INCOME : INfrastructure de gestion de COntexte Multi-Echelle pour l'Internet des Objets*. In Conférence francophone sur les architectures logicielles (CAL 2014), Paris, France, Juin 2014. (Cité en page 139.)

[Arcangeli 2014b] Jean-Paul Arcangeli, Sophie Chabridon, Denis Conan, Thierry Desprats, Romain Laborde, Sébastien Leriche, Léon Lim, Chantal Taconet, Raja

Boujbel, Samer Machara Marquez, Pierrick Marie and Sam Rottenberg. *Gestion de contexte multi-échelle pour l'Internet des objets*. In Ubimob'14 : 10èmes journées francophones Mobilité et Ubiquité, pages 1–8, Sophia-Antipolis, France, Juin 2014. (Cité en page 139.)

[Autili 2012] Marco Autili, Paola Inverardi, Patrizio Pelliccione and Massimo Tivoli. *Developing highly complex distributed systems : a software engineering perspective*. Journal of Internet Services and Applications, vol. 3, no. 1, pages 15–22, 2012. (Cité en page 14.)

[Bézivin 2001] J. Bézivin and O. Gerbé. *Towards a precise definition of the OMG/MDA framework*. In Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001, (ASE 2001), pages 273–280, Nov 2001. (Cité en page 31.)

[Blair 2009] G. Blair, N. Bencomo and R.B. France. *Models@ run.time*. Computer, vol. 42, no. 10, pages 22–27, Octobre 2009. (Cité en page 37.)

[Blair 2012] Gordon Blair and Paul Grace. *Emergent Middleware : Tackling the Interoperability Problem*. Internet Computing, IEEE, vol. 16, no. 1, pages 78–82, jan.-feb. 2012. (Cité en pages 15, 43 et 45.)

[Blanc 2005] Xavier Blanc. MDA en Action : Ingénierie logicielle guidée par les modèles. Edition Eyrolles. 2005. (Cité en page 31.)

[Boujbel 2013] Raja Boujbel, Sébastien Leriche and Jean-Paul Arcangeli. *A DSL for multi-scale and autonomic software deployment*. In ICSEA 2013, The Eighth International Conference on Software Engineering Advances, pages 291–296, Octobre 2013. (Cité en page 88.)

[Boujbel 2014] Raja Boujbel, Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Jean-Paul Arcangeli and Claire Lecocq. *MuScADeL : A Deployment DSL based on a Multiscale Characterization Framework*. In IEEE 38th Annual International Computers, Software and Applications Conference Workshops (COMPSAC 2014), pages 708–715, Juillet 2014. (Cité en page 139.)

[Bourbaki 1970] N. Bourbaki. Théorie des ensembles. Springer Berlin Heidelberg, 1970. (Cité en page 51.)

[Chabridon 2013] S. Chabridon, D. Conan, Z. Abid and C. Taconet. *Building ubiquitous QoC-aware applications through model-driven software engineering*. Science of Computer Programming, vol. 78, no. 10, pages 1912–1929, Octobre 2013. (Cité en page 37.)

[Chavalarias et al. 2009] David Chavalarias et al. *French Roadmap for complex Systems 2008-2009*, Mars 2009. (Cité en pages 4 et 14.)

[Chen 2012] Yen-Kuang Chen. *Challenges and opportunities of internet of things.* In Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific, pages 383 –388, 30 2012-feb. 2 2012. (Cité en page 21.)

[Conan 2014] Denis Conan and Léon Lim. *Distributed Event-Based System with Multi-scoping for Multiscalability.* In Proc. 9th Middleware Workshop on Middleware for Next Generation Internet Computing, Bordeaux, France, Décembre 2014. (Cité en page 99.)

[Dey 2001] A.K. Dey, G.D. Abowd and D. Salber. *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.* Special Issue on Context-Aware Computing in the Human-Computer Interaction Journal, vol. 16, no. 2–4, pages 97–166, December 2001. (Cité en page 43.)

[Ferry 2013] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin and Arnor Solberg. *Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems.* In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD '13, pages 887–894, Washington, DC, USA, Juin 2013. IEEE Computer Society. (Cité en page 37.)

[Fouquet 2014] François Fouquet, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau and Jean-Marc Jézéquel. *Kevoree Modeling Framework (KMF) : Efficient modeling techniques for runtime use.* CoRR Computing Research Repository, vol. abs/1405.6817, 2014. (Cité en page 37.)

[France 2007] Robert France and Bernhard Rumpe. *Model-driven Development of Complex Software : A Research Roadmap.* In Future of Software Engineering, 2007. FOSE '07, FOSE'07, pages 37–54, Washington, DC, USA, Mai 2007. IEEE Computer Society. (Cité en pages 24 et 36.)

[Franklin 2005] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy and Frederick Reiss. *Design Considerations for High Fan-in Systems : The HiFi Approach.* In In CIDR,Conference on Innovative Data Systems Research, pages 290–304, Asilomar, Californie, Janvier 2005. (Cité en page 47.)

[Gassara 2013] Amal Gassara, Ismael Bouassida Rodriguez and Mohamed Jmaiel. *Towards a Multi-scale Modeling for Architectural Deployment Based on Bigraphs.* In Khalil Drira, editeur, Software Architecture, volume 7957 of *Lecture Notes*

in Computer Science, pages 122–129. Springer Berlin Heidelberg, 2013. (Cité en page 37.)

[Gluhak 2011] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton and T. Razafindralambo. *A survey on facilities for experimental internet of things research.* Communications Magazine, IEEE, vol. 49, no. 11, pages 58 –67, november 2011. (Cité en page 22.)

[Götz 2014] Sebastian Götz, Nelly Bencomo and Robert France, editeurs. Proceedings of the 9th Workshop on Models@run.time (MRT 2014) co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, Septembre 2014. (Cité en page 30.)

[Hourcade 2009] JC Hourcade, R Saracco, Y Neuvo, W Wahlster and R Posch. *Future Internet 2020, Call for action by a high level visionary panel.* Report of the European Commission Information Society and Media, Brussels, Belgium, 2009. (Cité en page 82.)

[ISO/IEC/IEEE 2011] ISO/IEC/IEEE. *Systems and software engineering — Architecture description.* International Standard ISO/IEC/IEEE-42010 :2011, ISO/IEC/IEEE Joint Technical Committee, Décembre 2011. (Cité en page 42.)

[Jézéquel 2012] Jean-Marc Jézéquel, Benoit Combemale and Didier Vojtisek. Ingénierie Dirigée par les Modèles : des concepts à la pratique... Références sciences. Ellipses, Février 2012. (Cité en pages 28 et 31.)

[Jouault 2008] Frédéric Jouault, Freddy Allilaire, Jean Bézivin and Ivan Kurtev. *ATL : A model transformation tool.* Science of Computer Programming, vol. 72, no. 1–2, pages 31–39, 2008. Special Issue on Second issue of experimental software and toolkits (EST). (Cité en pages xiii, 34 et 35.)

[Kessis 2009] Mehdi Kessis, Claudia Roncancio and Alexandre Lefebvre. *DASIMA : A Flexible Management Middleware in Multi-Scale Contexts.* In Information Technology : New Generations, 2009. ITNG '09. Sixth International Conference on, pages 1390 –1396, april 2009. (Cité en pages 15, 16 et 46.)

[Machara Marquez 2013] Samer Machara Marquez, Sophie Chabridon and Chantal Taconet. *Trust-based Context Contract Models for the Internet of Things.* In Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC), pages 557–562, Vietri Sul Mare, Italy, Décembre 2013. (Cité en page 91.)

[Mell 2011] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg, USA, Septembre 2011. (Cité en page 22.)

[Miorandi 2012] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini and Imrich Chlamtac. *Internet of things : Vision, applications and research challenges*. Ad Hoc Networks, vol. 10, no. 7, pages 1497–1516, 2012. (Cité en page 21.)

[Morin 2009] Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey and Arnor Solberg. *Models@ Run.time to Support Dynamic Adaptation*. IEEE Computer, vol. 42, no. 10, pages 44–51, 2009. (Cité en page 37.)

[OMG 2002] OMG. *Meta Object Facility (MOF) Specification, v1.4*. formal/2002-04-03, Avril 2002. Object Management Group. (Cité en pages 31 et 63.)

[OMG 2008a] OMG. *MOF Model to Text Transformation Language, v1.0*. formal/2008-01-16, Janvier 2008. Object Management Group. (Cité en page 75.)

[OMG 2008b] OMG. *Software & Systems Process Engineering Metamodel (SPEM), v2.0*. formal/2008-04-01, Avril 2008. Object Management Group. (Cité en page 62.)

[OMG 2011] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, v1.1*. formal/2011-01-01, Janvier 2011. Object Management Group. (Cité en page 35.)

[OMG 2014] OMG. *OMG Meta Object Facility (MOF) Core Specification, v2.4.2*. formal/2014-04-03, Avril 2014. Object Management Group. (Cité en page 31.)

[Pons 2011] Pascal Pons and Matthieu Latapy. *Post-processing hierarchical community structures : Quality improvements and multi-scale view*. Theoretical Computer Science, vol. 412, no. 8- 10, pages 892–900, Mars 2011. (Cité en page 24.)

[Rottenberg 2012] Sam Rottenberg, Sébastien Leriche, Claire Lecocq and Chantal Taconet. *Vers une définition des systèmes répartis multi-échelle*. In Ubimob '12 : 8èmes journées francophones Mobilité et Ubiquité, 2012. (Cité en page 139.)

[Rottenberg 2014a] Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Claire Lecocq and Thierry Desprats. *MuSCa : A Multiscale Characterization Framework for Complex Distributed Systems*. In Federated Conference on Computer Science and Information Systems (FedCSIS 2014), pages 1687–1695, Warsaw, Poland, Septembre 2014. (Cité en page 139.)

[Rottenberg 2014b] Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Claire Lecocq and Thierry Desprats. *MuScA : A Multiscale Distributed Systems Scale-Awareness Framework*. In Conférence francophone sur les architectures logicielles (CAL 2014), Paris, France, Juin 2014. (Cité en page 139.)

[Sandhu 1992] Harjinder S. Sandhu and Songnian Zhou. *Cluster-based file replication in large-scale distributed systems*. In Proceedings of the ACM Sigmetrics Performance '92 Conference, SIGMETRICS '92/PERFORMANCE '92, pages 91–102, New York, NY, USA, Mai 1992. ACM. (Cité en page 4.)

[Satyanarayanan 1990] Mahadev Satyanarayanan. *Scalable, Secure, and Highly Available Distributed File Access*. IEEE Computer, vol. 23, no. 5, pages 9–18, Mai 1990. (Cité en page 4.)

[Satyanarayanan 2001] M. Satyanarayanan. *Pervasive Computing : Vision and Challenges*. IEEE Personal Communications, vol. 8, no. 4, pages 10–17, August 2001. (Cité en page 23.)

[Satyanarayanan 2009] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres and Nigel Davies. *The Case for VM-Based Cloudlets in Mobile Computing*. IEEE Pervasive Computing, vol. 8, pages 14–23, October-December 2009. (Cité en pages 18, 20 et 23.)

[Satyanarayanan 2011] Mahadev Satyanarayanan. *Mobile computing : the next decade*. SIGMOBILE Mobile Computing and Communications Review, vol. 15, pages 2–10, August 2011. (Cité en pages 15 et 18.)

[Schmidt 2006] Douglas C. Schmidt. *Guest Editor's Introduction : Model-Driven Engineering*. IEEE Computer, vol. 39, no. 2, pages 25–31, Février 2006. (Cité en pages 36 et 37.)

[Serral 2010] Estefanía Serral, Pedro Valderas and Vicente Pelechano. *Towards the Model Driven Development of context-aware pervasive systems*. Pervasive and Mobile Computing, vol. 6, no. 2, pages 254–280, Avril 2010. (Cité en page 37.)

[Tianfield 2008] H. Tianfield. *Fundamentals and architectures of Complex Distributed Systems*. In Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on, pages 2471–2475, Octobre 2008. (Cité en page 14.)

[van Steen 2012] Maarten van Steen, Guillaume Pierre and Spyros Voulgaris. *Challenges in very large distributed systems*. Journal of Internet Services and Applications, vol. 3, pages 59–66, 2012. 10.1007/s13174-011-0043-x. (Cité en page 15.)

[Warneke 2001] B. Warneke, M. Last, B. Liebowitz and K. S. J. Pister. *Smart Dust : Communicating with a Cubic-Millimeter Computer*. IEEE Computer, vol. 34, no. 1, pages 44–51, Janvier 2001. (Cité en page 20.)

[White 2012] Jules White, Brian Dougherty, Richard Schantz, Douglas C. Schmidt, Adam Porter and Angelo Corsaro. *R&D challenges and solutions for highly complex distributed systems : a middleware perspective*. Journal of Internet Services and Applications, vol. 3, no. 1, pages 5–13, 2012. (Cité en page 14.)