THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *30/01/2015* **par :**

Raja BOUJBEL

Déploiement de systèmes répartis multi-échelles : processus, langage et outils intergiciels

---

JURY

Jean-Paul ARCANGELI	Maître de Conférences HDR Université de Toulouse, UPS	Directeur
Jean-Paul BODEVEIX	Professeur Université de Toulouse, UPS	Examinateur
Amel BOUZEGHOUB	Professeur Institut Mines Télécom, TSP	Examinateur
Didier DONSEZ	Professeur Université Grenoble Alpes, UJF	Rapporteur
Sébastien LERICHE	Maître de Conférences Université de Toulouse, ENAC	Co-Directeur
Phillipe ROOSE	Maître de Conférences HDR Université de Pau et des Pays de l'Adour, IUT de Bayonne et du Pays Basque	Rapporteur

---

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
IRIT
Directeur(s) de Thèse :
Jean-Paul ARCANGELI et *Sébastien LERICHE*
Rapporteurs :
Didier DONSEZ et *Philippe ROOSE*

Résumé

Avec la multiplication des objets connectés, les systèmes multi-échelles (qui vont de l'Internet des objets au Cloud, en passant par des machines personnelles, des smartphones, des tablettes, etc.) sont de plus en plus répandus. Ces systèmes sont fortement répartis, hétérogènes, dynamiques et ouverts (apparition et disparition d'appareils). Ils peuvent être composés de centaines de composants logiciels déployés sur des milliers d'appareils.

Le déploiement est un processus complexe qui a pour objectif la mise à disposition puis le maintien en condition opérationnelle d'un système logiciel. Pour les systèmes multi-échelles, l'expression du plan de déploiement (association entre les composants logiciels et les appareils) ainsi que la réalisation et la gestion 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 forcément connu à l'avance. Le déploiement n'est, de plus, pas réservé à un ingénieur spécialiste mais, dans le cadre d'applications grand public, certains utilisateurs peuvent aussi être parties prenantes.

Ainsi, pour assurer le déploiement, il est nécessaire de disposer d'une part de moyens d'expression et d'abstraction, et d'autre part, de supports pour la réalisation automatisée (construction du plan, mise en œuvre du plan puis maintien en condition opérationnelle du système). Or, les solutions de déploiement actuelles sont globalement inadaptées ou incomplètes : elles ne prennent pas en compte l'ensemble des caractéristiques des systèmes multi-échelles. Le contexte multi-échelle nécessite, par ailleurs, de nouvelles formes de spécification relatives à des parties du domaine de déploiement identifiées selon différents points de vue ou échelles (géographiques, sociaux, etc.).

L'objectif de cette thèse est d'étudier et de proposer des solutions pour le déploiement de systèmes répartis multi-échelles. Nous proposons tout d'abord une mise à jour du vocabulaire relatif au déploiement, ainsi qu'un état de l'art sur le déploiement automatique des systèmes logiciels répartis. Le reste de la contribution réside dans la proposition :

— d'un processus complet pour le déploiement autonomique de systèmes multi-échelles (qui s'appuie sur les deux éléments décrits ci-dessous) ;
— d'un langage dédié (DSL), MuScADeL, qui simplifie la tâche du concepteur du déploiement et permet d'exprimer les contraintes propres aux composants logiciels du système à déployer, les choix de conception, les propriétés relatives aux aspects multi-échelles, ainsi que de spécifier les sondes nécessaires à la perception de l'état du domaine de déploiement ;
— d'un middleware, MuScADeM, qui assure la génération automatique d'un plan de déploiement en fonction de l'état du domaine, sa réalisation puis le main-

tien en condition opérationnelle du système. L'infrastructure de déploiement est décentralisée, et des composants autonomes sont capables de réagir à des situations d'adaptation (variations de l'état du domaine) selon les principes de l'informatique autonomique.

En pratique, un plugin Eclipse permet l'édition de propriétés exprimées en MuScADeL, avec plusieurs niveaux de vérification. Il permet aussi de déclencher la génération d'un plan de déploiement.

En matière de réalisation, notre solution s'appuie sur OSGi, ce qui permet d'assurer les activités d'installation et d'activation, mais aussi des activités tel que l'arrêt, la mise à jour, etc. L'utilisation d'un framework OSGi permet à MuScADeM de s'affranchir des problèmes d'hétérogénéité.

Ces travaux ont fait l'objet de plusieurs publications, et les solutions sont expérimentées dans le cadre du projet INCOME financé par l'Agence Nationale de la Recherche.

Remerciements

« This is my thesis
There are many like it
but this one is mine »
Raja
inspired from a mantis shrimp
inspired from Full Metal Jacket

Certains diraient que c'est la partie la plus difficile à écrire. Certains que c'est la plus facile. Certains ne communiqueraient pas sur ce point. Certains commenceront par ce récapitulatif.

Mais finalement, certains diront qu'après avoir fini de l'écrire, c'était amusant de le faire.

Je voudrais tout d'abord remercier les membres du jury. Mes rapporteurs, Didier Donsez et Philippe Roose pour leur travail de relecture, malgré le court délai. Mes examinateurs, Amel Bouzeghoub et Jean-Paul Bodeveix pour avoir accepté d'évaluer mon travail de thèse.

Mais le jury n'est pas encore complet, il manque mes directeurs. Je remercie Jean-Paul et Sébastien pour m'avoir permis d'effectuer cette thèse, la mener à bout, jusqu'à ce jour de soutenance. Ce n'est pas la première, mais celle-là touche à sa fin et se concrétise. Jean-Paul dirait que ma seule envie serait qu'il me laisse tranquille après ces quelques semaines. Au contraire, étant en fin de contrat, je vais me sentir trop tranquille. Au moins Sébastien n'aura plus à essayer de nous faire comprendre. J'ai aimé réaliser cette thèse, et c'est grâce à vous, votre direction, votre enthousiasme, nos brainstormings… Et c'est aussi grâce à vous que j'ai découvert Toulouse, qui aujourd'hui me fait hésiter à retourner à Paris.

Mais Toulouse n'aurait peut-être pas été la même si je n'avais atterri dans l'équipe SMAC [1]. J'y suis arrivée à la période la plus dense en doctorants, ce qui m'a permis de rencontrer plein de personnes, personnalités différentes. Les pauses cafés, auxquelles certains permanents assistaient, étaient des moments de détente, d'échanges parfois productifs, parfois houleux (n'est-ce pas Tom et Arcady ?), parfois perchés (j'en suis encore à être surprise par le contenu de Minecraft), parfois incompréhensibles pour certains (tu veux une explication Valérian ?) mais toujours agréables (pas toujours pour ceux qui entendent le brouhaha, mais bon). Il faut faire attention à faire perdurer cette coutume, même les non-buveurs de café. Mais les pauses cafés ne se sont pas prises que dans le couloir. Le bureau d'à côté a aussi été un lieu de discussions de toutes sortes, et parfois de refuge (non je ne me moque pas). Je ne veux pas faire de liste de noms (en plus on en oublie à tous les coups). Merci à

1. Je ne peux pas l'affirmer, je n'ai pas eu l'opportunité de lancer un dé pour voir ce qui se passerait dans les autres dimensions.

tous les docteurs et doctorants pour leur bonne humeur, leurs échanges permanents, que ce soit à l'IRIT ou en dehors. Merci à Valérian d'avoir été une des causes de cette dynamique qui existe aujourd'hui et qui fait du bien quand on arrive fraîchement dans l'équipe. Ta jovialité et consternante capacité à discuter de tout et de rien (mais principalement de toi, il ne faut pas perdre le nord quand même) a beaucoup aidé à pas mal de moments. Merci à Luc pour tes conseils, ta détermination, ta compagnie. Merci à Nicolas pour les nombreuses conversations et ton soutien {\m/}. Merci à Christine pour votre bonne humeur, votre sensibilité à la taquinerie, votre côté maman. En fait merci à tout le monde pour tous, les discussions intéressantes, constructives, déterministes, déterminées, (bonnes ou mauvaises) conseillères, controversées, futiles, complotistes… et pour votre confiance.

Je voudrais aussi remercier mes anciens collègues, et plus précisément Louis, David, et François. Malgré tous les déboires qu'il y a eu, ce fût une excellente période en votre compagnie. C'est avec vous que je me suis forgée mon âme d'informaticienne.

Sans oublier les parisiennes, Zouhour et Virginie. Malgré la distance, toujours présentes dans les bons et mauvais moments. Merci. Et le nouveau toulousain, Vincent. La rédaction conjointe, ça a porté ses fruits.

Merci aussi à ma famille pour leur soutien, mes frère et sœur, Rym et Karim, leurs conjoints, Nébil et Samar, et surtout mes parents Béchir et Afifa. Vous avez laissé partir une jeunette de 18 ans et comme dirait tonton Mehdi, maintenant on voit où ça mène, à cette soutenance. Je vous remercie de m'avoir donné l'opportunité de choisir la voie qui me passionne.

The last but not the least, Guillaume. Merci de m'avoir fait découvrir l'informatique, d'avoir participé à faire de moi l'informaticienne que je suis. Si j'en suis là aujourd'hui, c'est aussi grâce à toi. Depuis le temps, on est toujours là, contre vents et marées. Merci pour ton soutien, ta geekerie, ta passion, tes conseils, ta patience, ton calme… toi.

Table des matières

Liste des figures

Liste des tableaux

Liste des listings

1 Introduction

1.1 Contexte

« *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.* » [Weiser, 1991]. En 1988, Mark Weiser propose le terme « informatique ubiquitaire » pour désigner sa vision de l'informatique, où l'utilisateur ne se servirait pas seulement d'un ordinateur, mais où les technologies environnantes serviraient l'utilisateur.

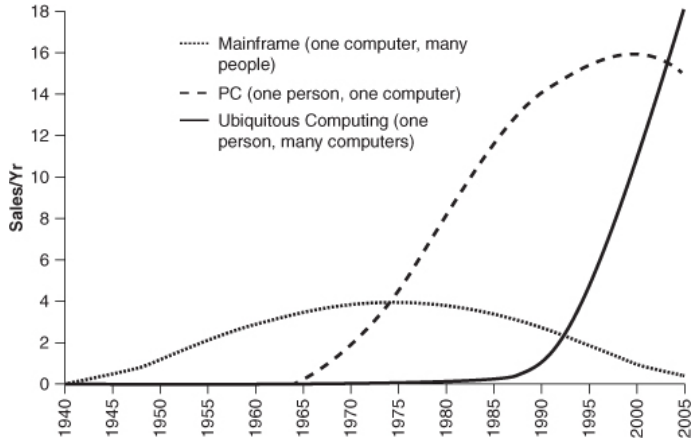


Figure 1.1 – Les tendances majeures de l'Informatique [Wei]

Plus de 20 ans plus tard, nous possédons plusieurs terminaux, nous sommes passés d'un paradigme où un ordinateur était utilisé par plusieurs personnes à plusieurs appareils utilisés par une seule personne (cf. figure 1.1). L'informatique ubiquitaire a mûri – en passant par les stades de la connectivité, de la perception de l'environnement et de l'intelligence – sans que toutes les problématiques inhérentes ne soient réglées [Ferscha, 2012]. Les avancées technologiques – la miniaturisation du matériel, le gain en autonomie, les réseaux de communication – ont permis cette prolifération et la diversité des appareils connectés. En parallèle, de nouvelles technologies logicielles et applications se sont développées pour s'adapter à ce nouveau paradigme.

Nous sommes passés d'un monde où les seules sources d'information était celles diffusées par les grands médias (radio, télévision) à un monde où l'information est diffusée de manière horizontale (Web 2.0, Twitter, blogs) dans lequel l'utilisateur recherche l'information dont il a besoin. Puis, peu à peu, avec le développement de l'informatique ubiquitaire,

l'utilisateur n'a plus besoin d'effectuer cette recherche, mais l'information vient à lui selon ses préférences. Par exemple, il y a 30 ans, une personne qui cherchait un restaurant dans une ville étrangère devait trouver l'information dans le journal ou la critique gastronomique de la ville, à condition de parler la langue. Il y a 15 ans, il lui était déjà possible de faire une recherche sur Internet sur des sites spécialisés en gastronomie. Aujourd'hui, il lui suffit de se promener en ville à l'heure du déjeuner pour que son smartphone lui dresse une liste de restaurants dans le quartier selon ses préférences (culinaires, prix, services, etc.) en sélectionnant les meilleurs à partir des avis des clients et des critiques, et lui indique le moyen de s'y rendre.

Cette facilité d'utilisation des technologies est actuellement possible grâce à une infrastructure centralisée. Un site de recommandation agrège l'information, les restaurants y sont inscrits, les clients y partagent leurs avis, et les employés du site doivent valider et organiser l'information. Cette centralisation n'offre pas au restaurateur la maîtrise du contenu concernant son établissement. De plus, la fraîcheur de l'information (par exemple fermeture définitive ou temporaire du restaurant) n'est pas garantie. Nous pouvons imaginer un système complètement décentralisé, où à l'heure du déjeuner, les différents restaurants envoient directement à l'utilisateur demandeur le menu avec le plat du jour, les offres promotionnelles (par exemple, adaptées à la météo), le nombre de tables restantes, etc. Le restaurateur maîtrise alors les informations envoyées, leur justesse dans le temps, sans impliquer d'autres ressources humaines externes, et on évite aussi une certaine latence. Ainsi, la production et la consommation d'information sont décentralisées.

1.2 Problématique

Ces nouvelles applications de l'informatique ubiquitaire doivent être réparties sur différents types d'appareils : celui de l'utilisateur, celui du restaurateur, celui qui effectue les sauvegardes, etc. De plus, pour chaque nouvel utilisateur, l'application doit être disponible et prête à l'utilisation. Pour cela, un système doit la mettre à disposition sur tous les appareils. Un tel système est un système de déploiement. Son but est de rendre un logiciel disponible à l'utilisation et, ensuite, de le maintenir opérationnel.

Les solutions traditionnelles de déploiement privilégient un mode centralisé dans lequel un opérateur humain réalise les différentes tâches du déploiement. Ces solutions répondent principalement aux problèmes du déploiement dans un environnement hiérarchique plus ou moins statique, comme un réseau d'entreprise ou une grille de calcul. Le déploiement est piloté depuis un ordinateur, de manière centralisée, de l'installation à la mise à jour et à la désinstallation. Pour déployer, l'opérateur doit posséder une double expertise, sur l'application à déployer et sur les opérations de déploiement. Il lui faut indiquer sur quels appareils les composants du système doivent être déployés, initier le déploiement et être capable de réagir en cas de panne ou d'apparition d'un nouvel appareil.

Or, les nouvelles applications de l'informatique ubiquitaire qui vont de l'Internet des objets [Atzori et al., 2010] au Cloud, en passant par les machines personnelles, les smartphones, les tablettes, etc. sont fortement réparties et hétérogènes, décentralisées, dynamiques et ouvertes. Elles peuvent être composés de milliers de composants à déployer sur des centaines ou des milliers d'appareils, qui ne sont pas entièrement connus à l'avance du fait de l'apparition et la disparition d'appareils. Dans ce cadre, la définition et la réalisation des opérations de déploiement, ainsi que leur supervision, sont des tâches humainement dif-

ficiles voire impossibles. De plus, une solution centralisée est peu envisageable dans un tel contexte de nombre et de répartition. Pour répondre à ces nouveaux problèmes en matière de déploiement, de nouvelles formes de processus, de nouveaux moyens d'expression, de nouvelles techniques de réalisation doivent être proposés.

1.3 INCOME et les systèmes multi-échelles

De nos jours, de plus en plus de systèmes combinent le concept d'ordinateur invisible et ubiquitaire intégré dans l'environnement physique [Weiser, 1999], l'Internet des Objets et la mobilité des appareils des utilisateurs. À cause des limitations matérielles et logicielles, des ressources distantes sont nécessaires et peuvent être fournies par les infrastructures de cloud. Ces systèmes qui sont distribués sur les objets intelligents, les passerelles des réseaux de capteurs et des infrastructures de cloud, les appareils mobiles et les serveurs fixes, sont appelés « systèmes répartis multi-échelles » [Kessis et al., 2009, Flinn, 2012, van Steen et al., 2012]. Ces systèmes sont complexes par nature, de par le nombre, l'hétérogénéité, la dynamique et les différents niveaux d'organisation et de structure qu'ils contiennent. L'objet du travail de thèse de S. Rottenberg [Rottenberg, 2015] (voir également [Rottenberg et al., 2014]) est d'étudier le concept de système multi-échelle et de proposer une approche à base d'ingénierie dirigée par les modèles afin de faciliter leur conception et, en particulier, de permettre au développeur de travailler sur des vues réduites et simplifiées avec bon niveau d'abstraction. Il faut noter que le concept de « multi-échelle » diffère de celui de « grande échelle » qui a un sens quantitatif.

Ce travail de thèse s'inscrit dans le cadre du projet INCOME (INfrastructure de gestion de COntexte Multi-Échelle pour l'Internet des Objets) [INC, Arcangeli et al., 2012c], financé par l'Agence Nationale de la Recherche (ANR). Le projet INCOME cible l'informatique sensible au contexte dans le cadre des systèmes multi-échelles. Pour la sensibilité au contexte, la capacité à gérer les informations de contexte (collecte, agrégation, filtrage, transformation, acheminement, etc.) sont primordiales. L'objectif du projet INCOME de concevoir des solutions pour la gestion de contexte répartie multi-échelle [Arcangeli et al., 2012a]. Outre, l'étude du concept de système multi-échelle, le projet est divisé en plusieurs volets :

- un volet fonctionnel : collecte, traitement et acheminement des informations de contexte multi-échelle ;
- un volet extra-fonctionnel : qualité de contexte et respect de la vie privée ;
- un volet opérationnel : déploiement du gestionnaire de contexte multi-échelle.

Dans le cadre du projet INCOME, nous avons défini un scénario d'application utilisatrice d'un gestionnaire de contexte multi-échelle, appelé Mobilité MultiModale Multi-échelle (4ME). Les utilisateurs de 4ME peuvent se rendre à une destination en utilisant les différents moyens de transports personnels (voitures, vélo, etc.), partagés ou collectifs (métro, bus, vélo en libre service, etc.). 4ME offre diverses fonctionnalités et tire profit des installations urbaines pour offrir ses services. Un citadin peut utiliser 4ME pour avoir des informations concernant son trajet quotidien, alors qu'un touriste peut utiliser 4ME pour découvrir et être guidé dans la ville. Dans les deux cas, différents moyens permettent de connaître en temps-réel l'état des réseaux de transport (temps d'attente, disponibilité de vélos ou emplacements vides, etc.) et du trafic routier (chemin préférentiel, perturbations, etc.).

L'application 4ME utilise un gestionnaire de contexte pour traiter les informations de contexte dont elle a besoin. Ce gestionnaire de contexte est un système de compo-

sants répartis qui doivent être déployés sur différents appareils : les appareils mobiles pour l'interaction directe avec l'utilisateur (interface graphique, information de localisation, préférences, etc.), les appareils producteurs d'information (borne de vélo, bus, arrêt de bus, réseau du métro, etc.), les serveurs qui traitent ces informations de contexte avant de les transmettre à l'application 4ME, etc. La figure 1.2 représente les différents appareils et services de 4ME.

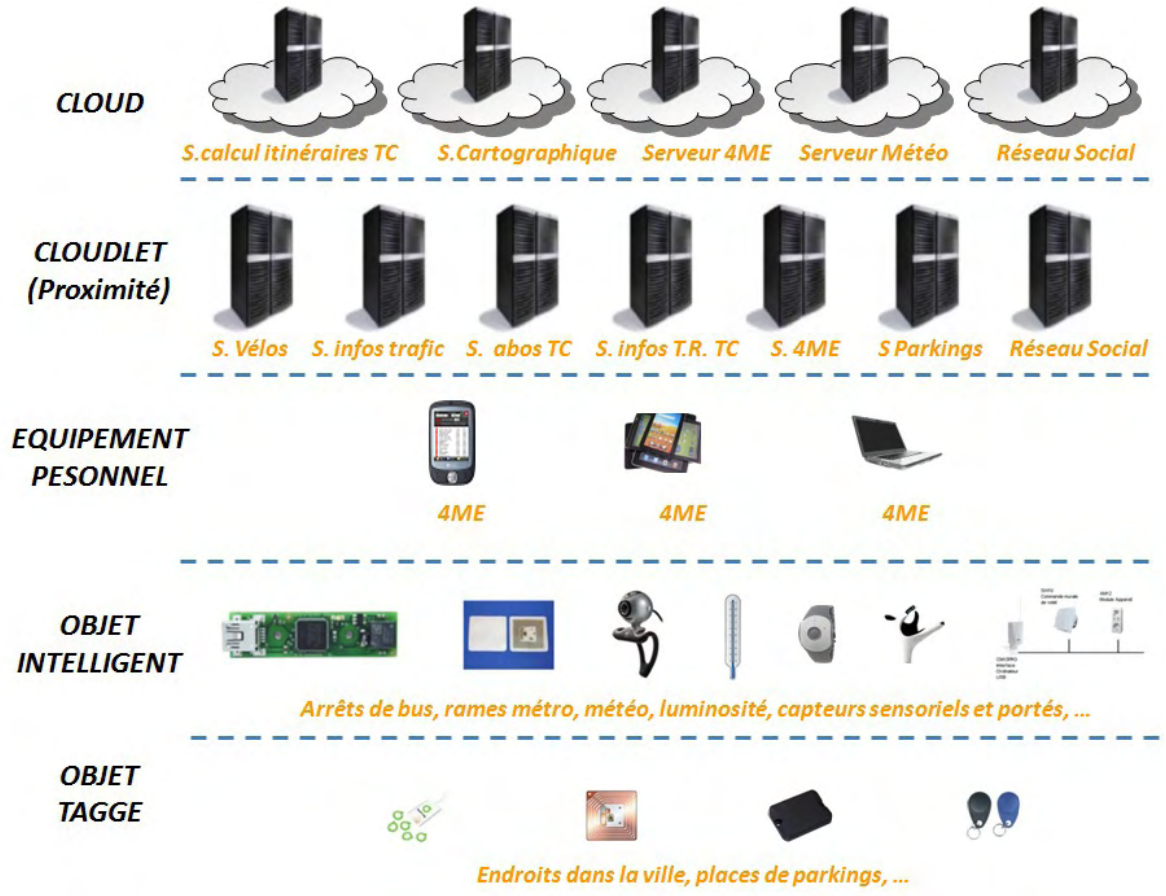


Figure 1.2 – Répartition des éléments de 4ME, de l'IoT au Cloud

4ME et le gestionnaire de contexte sont des systèmes multi-échelles. Leur déploiement sur les différents appareils hôtes est une tâche complexe, en particulier du fait de la multitude de composants à déployer et de la dynamique de l'environnement sur lequel déployer.

1.4 Contribution

Les solutions de déploiement actuelles sont globalement inadaptées ou incomplètes, au regard de l'ensemble des caractéristiques des systèmes multi-échelles. En particulier, le contexte multi-échelle demande de nouvelles formes de spécification relativement aux échelles (géographiques, sociales, etc.).

Nous proposons ici une solution générique pour le déploiement des systèmes multi-échelles. Cette solution permet d'exprimer simplement des propriétés de déploiement et de répondre à la dynamique et l'ouverture de l'environnement de manière automatique et

autonomique. Notre contribution réside en la définition :

— d'un processus complet pour le déploiement autonomique de systèmes multi-échelles (qui s'appuie sur les deux éléments décrits ci-dessous) ;
— d'un langage dédié (DSL), MuScADeL, qui simplifie la tâche du concepteur du déploiement et permet d'exprimer les contraintes propres aux composants logiciels du système à déployer, les choix de conception, les propriétés relatives aux aspects multi-échelles, ainsi que de spécifier les sondes nécessaires à l'acquisition du contexte de déploiement ;
— d'un middleware, MuScADeM, qui assure la génération automatique d'un plan de déploiement en fonction du contexte, puis sa réalisation et le maintien en condition opérationnelle du système. L'infrastructure de déploiement est décentralisée, et des éléments autonomes sont capables de réagir à des situations d'adaptation selon les principes de l'informatique autonomique.

1.5 Plan du mémoire

Ce manuscrit s'organise comme suit. Le chapitre 2 présente une mise à jour de la terminologie du déploiement et analyse la problématique du déploiement des systèmes répartis multi-échelles. Le chapitre 3 dresse ensuite un état de l'art sur l'automatisation du déploiement. Dans le chapitre 4 un processus pour le déploiement automatique et autonomique des systèmes répartis multi-échelles est proposé. Le chapitre 5 présente le langage MuScADeL pour la spécification des propriétés de déploiement. Le chapitre 6 définit une architecture pour l'automatisation du déploiement, et une formalisation des propriétés de déploiement sous la forme de contraintes. Dans le chapitre 7, les situations d'adaptation dynamique sont analysées et des éléments d'architecture pour le déploiement autonomique sont proposés. Le chapitre 8 expose les choix technologiques et la réalisation d'un prototype selon l'architecture définie, ainsi que des éléments de validation. Enfin, une conclusion est donnée dans le chapitre 9 et des perspectives sont discutées.

2 Définitions et analyse du problème

Problématique

Le déploiement de logiciel est un processus complexe qui a pour objectif la mise à disposition puis le maintien en condition opérationnelle du logiciel. Les définitions de référence en matière de déploiement considèrent principalement les activités de réalisation. Elles doivent être actualisées et complétées afin de mieux prendre en compte les aspects répartis, ubiquitaires, ouverts et dynamiques des systèmes logiciels actuels. Dans le cadre des systèmes logiciels répartis multi-échelles, les solutions de déploiement doivent satisfaire un certain nombre d'exigences en lien avec les problèmes d'hétérogénéité, de nombre, de dynamique et d'ouverture.

2.1 Introduction

Le déploiement de logiciel est un processus complexe qui a pour objectif la mise à disposition puis le maintien en condition opérationnelle du logiciel. Le déploiement suit la production du logiciel, mais aussi l'accompagne, et coordonne un ensemble d'activités liées comme le dépôt du logiciel, l'installation, l'activation, la mise à jour, l'adaptation, la suppression, etc.

Les produits logiciels actuels ne sont plus monolithiques mais composés d'un ensemble de « composants logiciels » assemblés en un « système logiciel » et opérant ensemble. Le terme « composant » fait référence à un élément d'un système dans un contexte de modularité, ou plus spécifiquement à une unité de composition dont les interfaces sont définies contractuellement et qui est sujette à composition par une tierce partie [Szyperski, 2002, Crnkovic et al., 2011]. Dans ce cas, les interfaces spécifient à la fois les fonctions ou les services fournis par le composant et ceux que le composant requiert d'autres composants ou de son environnement. De nombreux modèles de composants logiciels existent, tel que JavaBeans [Ora2], Corba Component Model [Obj2], ou Fractal [Bruneton et al., 2006, OW2]. Les technologies à base de composants facilitent le déploiement (entre autres choses, les composants peuvent fournir des interfaces dédiées à l'administration et à la configuration) et les composants peuvent être déployés indépendamment les uns des autres.

Aujourd'hui, l'usage de composants logiciels comme unité d'empaquetage, de transfert et de déploiement est devenu une pratique courante. Le déploiement d'un unique composant ne pose pas de problème majeur, mais le déploiement d'un système entier, donc la coordination du déploiement de ses composants est plus difficile. La répartition est l'un des aspects de ce problème, les systèmes logiciels étant communément répartis sur différentes machines ou appareils en réseau.

Notre compréhension du déploiement de logiciel se base sur les définitions données dans deux papiers de référence par Carzaniga *et al.* [Carzaniga et al., 1998] et Dearle [Dearle, 2007]. Des concepts génériques ont aussi été spécifiés par l'OMG [Obj1] dans la spécification D&C [Obj3]. Carzaniga *et al.* analysent les problèmes et les défis du déploiement de logiciel, et proposent un cadre de caractérisation pour les technologies de déploiement [Carzaniga et al., 1998]. Les principaux problèmes identifiés sont la prise en compte des changements, les dépendances entre composants, la distribution de contenu, la coordination entre le déploiement et l'utilisation du logiciel, l'hétérogénéité, la sécurité, la flexibilité du processus de déploiement et l'intégration avec Internet. Le cadre de caractérisation proposé est basé sur quatre critères : la couverture du processus de déploiement, la flexibilité du processus de déploiement, la coordination inter-processus et le support de la modélisation. En se basant sur ce cadre de caractérisation, les auteurs ont analysé un ensemble de technologies : installeurs, gestionnaires de paquet, applications de gestion de systèmes logiciels, technologies de distribution de contenu et des standards pour la description de systèmes logiciels.

Dearle donne une vue d'ensemble du déploiement de logiciel et identifie certains problèmes comme l'établissement des liens entre composants, l'utilisation de conteneurs et l'inversion du contrôle, la réflexivité et l'usage de metadonnées [Dearle, 2007]. Puis, il examine différents problèmes et leurs possibles solutions : il se concentre sur la granularité des conteneurs, le déploiement distribué, le middleware, l'adaptation dynamique et l'autonomie, puis la spécification architecturale. En particulier, Dearle discute du besoin d'autonomie dans le déploiement et en souligne la complexité potentielle.

Dans la première partie de ce chapitre, nous proposons d'actualiser et de compléter les définitions de Carzaniga *et al.* et de Dearle, dans le but de mieux prendre en compte les aspects répartis, ouverts et dynamiques des systèmes informatiques actuels. Dans la seconde partie, nous nous concentrons sur le problème du déploiement des systèmes logiciels répartis multi-échelles, que nous analysons afin d'en extraire un certain nombre d'exigences auxquelles une solution de déploiement doit répondre.

2.2 Définitions

2.2.1 Rôles

Dans le processus de déploiement, les parties prenantes peuvent jouer différents rôles. La plupart des auteurs considèrent un rôle unique généralement appelé « gestionnaire du déploiement ». Dearle, lui, fait une distinction entre « producteur du logiciel » et « déployeur du logiciel ». Dans cette thèse, nous raffinons ce dernier rôle en deux, qui sont « concepteur du déploiement » et « opérateur du déploiement », relatifs à des activités et à des compétences différentes. De plus, nous considérons deux rôles additionnels : « administrateur système » et « utilisateur du logiciel », chacun pouvant prendre une part active au

déploiement et ayant ses propres exigences.

Ainsi, nous définissons les rôles suivants :

— **Producteur du logiciel**. c'est le créateur du système logiciel ; il fournit aussi l'installeur du logiciel. Il agit en amont de la distribution du logiciel mais il doit prendre en compte certaines exigences en matière de déploiement.
— **Concepteur du déploiement**. Il exprime des commandes de déploiement ou bien seulement une spécification du déploiement (c'est-à-dire un ensemble de propriétés que les opérations de déploiement doivent respecter). Il peut utiliser un framework ou un langage dédié pour le faire.
— **Opérateur du déploiement**. Il est responsable de la supervision et de la réalisation du déploiement.
— **Administrateur système**. Il gère les ressources d'un appareil (concerné par le déploiement).
— **Utilisateur du logiciel**. C'est l'utilisateur final du logiciel déployé. Il agit en aval mais peut exprimer certaines exigences et préférences en matière de déploiement.

Un même acteur peut jouer plusieurs rôles dans le processus de déploiement. Par exemple, le propriétaire d'un smartphone peut être administrateur système (de son appareil), concepteur et opérateur du déploiement, ainsi qu'utilisateur du logiciel déployé. Inversement, plusieurs acteurs peuvent joueur le même rôle.

2.2.2 Répartition

Le déploiement implique deux catégories de sites [Carzaniga et al., 1998] : un « site producteur » qui héberge les éléments logiciels ainsi que les procédures d'installation, et un « site consommateur » qui est la cible du déploiement, sur lequel un élément logiciel est destiné à être exécuté.

Le déploiement d'un système peut être réalisé sur plusieurs sites consommateurs. L'ensemble de ces sites constitue un « domaine de déploiement », sur lequel il faudra décrire la répartition des composants, ce qui est appelé un « plan de déploiement ».

— **Domaine de déploiement**. Le domaine de déploiement est un ensemble d'appareils – ou machines – (sites de déploiement) connectés à un réseau de communication, qui hébergent un élément du système logiciel.
— **Plan de déploiement**. Le plan de déploiement est une correspondance entre les composants du système logiciel et les appareils du domaine de déploiement. Il contient souvent d'autres informations sur la configuration et les dépendances entre composants.

Un domaine de déploiement n'est pas obligatoirement statique et stable. Au contraire, de plus en plus d'applications s'exécutent dans des environnements dynamiques, mobiles et ouverts. Les domaines varient donc avec le temps au gré des mises en route ou des arrêts des appareils, ou encore des apparitions ou des disparitions de ceux-ci.

2.2.3 Activités

Pour Carzaniga *et al.*, le déploiement est une étape essentielle dans le cycle de vie du logiciel. Il recouvre l'ensemble des activités qui rendent un système logiciel disponible pour

l'utilisation : le dépôt (à la fin du processus de production), l'installation dans l'environnement d'exécution, l'activation, la désactivation, la mise à jour et le retrait des composants [Carzaniga et al., 1998].

Pour Dearle, le déploiement est une tâche de post-production qui peut être définie comme l'ensemble des processus qui opèrent entre l'acquisition et l'exécution du logiciel, et qui consistent en la conduite d'un certain nombre d'activités liées [Dearle, 2007].

Nous définissons le **déploiement de logiciel** comme un **processus qui organise et orchestre un ensemble d'activités ayant pour but de rendre le logiciel disponible à l'utilisation et de le maintenir à jour et opérationnel**. Certaines d'entre elles s'effectuent avant ou après l'exécution du logiciel (par exemple, dépôt ou retrait du logiciel sur le site producteur ou désinstallation du logiciel sur le site consommateur), alors que d'autres interviennent lorsque le logiciel est en cours d'exécution sur le(s) site(s) consommateur(s).

Il n'y a pas de consensus sur la définition du contenu de chaque activité ni sur leur nommage. En prenant pour référence Carzaniga *et al.* et Dearle, nous proposons pour cette thèse les noms et définitions suivants :

— **Dépôt**. Le dépôt concerne toutes les opérations nécessaires pour la préparation du(des) composant(s) logiciel(s) pour l'assemblage et la distribution (assemblage en paquetage contenant suffisamment de métadonnées pour décrire les ressources dont dépend le logiciel).
— **Installation**. L'installation est la première « intégration » du(des) composant(s) logiciel(s) sur le(s) site(s) consommateur(s). Elle requiert que le(s) composant(s) logiciel(s) soient transférés (distribution) et configurés afin de le(s) préparer à l'activation.
— **Activation**. L'activation couvre toutes les opérations requises pour démarrer le système logiciel ou installer des déclencheurs qui vont lancer le système logiciel à un moment donné. Pour les systèmes logiciels complexes, il est possible que d'autres services et processus aient aussi besoin d'être démarrés.
— **Désactivation**. La désactivation est l'inverse de l'activation, donc couvre toutes les opérations requises pour arrêter le système logiciel. Il se peut que des dépendances doivent être prises en compte.
— **Désinstallation**. La désinstallation (opération inverse de l'installation) supprime le(s) composant(s) logiciel(s) du(des) site(s) consommateur(s).
— **Retrait**. Le retrait est l'ensemble des opérations effectuées sur le site producteur, par le producteur du logiciel, pour marquer le logiciel comme étant obsolète. La principale conséquence du retrait du logiciel est qu'aucune nouvelle version ne sera produite (cependant, les versions courantes et précédentes peuvent rester utilisables).

Après l'installation, des opérations sont nécessaires pour faire évoluer le logiciel déployé. Pour les systèmes à base de composants, il existe des activités qui agissent localement au niveau du composant (mise à jour et adaptation) et d'autres qui agissent globalement sur le système (reconfiguration et redistribution).

— **Mise à jour**. La mise à jour est déclenchée par le dépôt d'une nouvelle version d'un composant sur le site producteur. Elle consiste à remplacer un composant par cette nouvelle version. Elle est similaire à l'installation, mais généralement moins complexe car la plupart des dépendances ont déjà été résolues.
— **Adaptation**. L'adaptation est déclenchée par un changement dans l'environnement du composant : environnement d'exécution, requête de l'opérateur ou de l'utilisateur,

etc. Dans ce cas, le logiciel doit être adapté afin de rester opérationnel. L'adaptation peut consister à remplacer un composant par un autre (comme pour une mise à jour) ou, plus simplement, à modifier des paramètres de configuration internes.

— **Reconfiguration**. La reconfiguration modifie globalement l'organisation du système de composants dans sa structure logique, c'est-à-dire les paramètres de configuration des liens entre les composants.

— **Redistribution**. La redistribution modifie la distribution physique du système c'est-à-dire le plan de déploiement. La redistribution peut par exemple, être requise lors d'un changement dans la topologie du réseau. Cette activité consiste à déplacer des composants tout en préservant la fonctionnalité du système. Le terme « redéploiement » est parfois utilisé comme un synonyme de redistribution.

2.2.4 Ordonnancement des activités

Il existe un ordre logique et des relations de précédence entre les différentes activités de déploiement. La figure 2.1 présente un ordre standard entre ces activités. On notera que le retrait d'un composant n'est pas forcément précédé par sa désinstallation : en effet, il peut devenir obsolète (retiré du site producteur) sans avoir été désinstallé (du site consommateur).

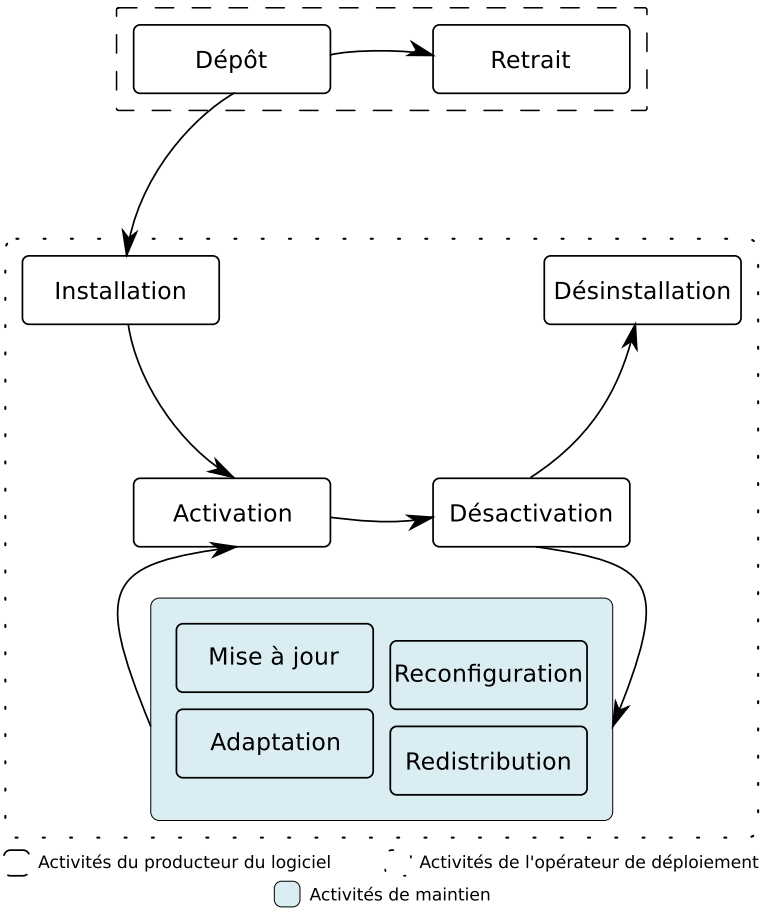


Figure 2.1 – Ordonnancement des activités de déploiement

Dans le cadre du déploiement, un composant a trois états possibles : *déployable*, *inactif*

et *actif*. Il est *déployable* lorsque le producteur du logiciel a effectué toutes les opérations nécessaires afin qu'il soit apte à être déployé. Il est *inactif* lorsque le logiciel est disponible à l'utilisation mais pas encore exécuté. Enfin, il est *actif* lorsqu'il est en cours d'exécution.

La machine à états représentée à la figure 2.2 montre comment les activités impactent l'état d'un composant et à quel moment elles peuvent intervenir. La mise à jour suppose une désactivation au préalable du composant. L'adaptation peut intervenir lors de l'état actif, mais dans ce cas, elle est limitée au changement de paramètres configuration (unitaires). Pour simplifier, l'opération de retrait n'est pas représentée : elle n'impacte pas l'état du composant logiciel déployé à part qu'il ne peut plus être mis à jour.

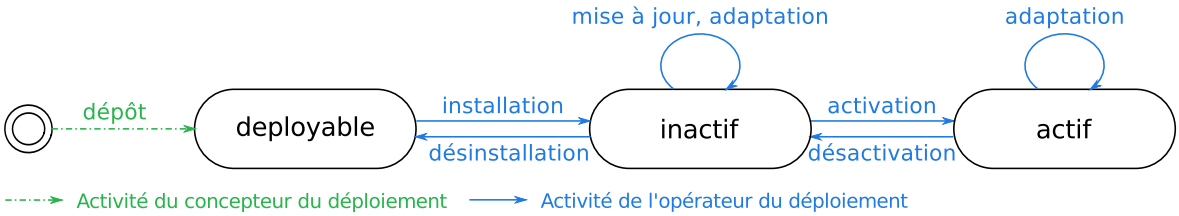


Figure 2.2 – Impact des activités de déploiement sur l'état du logiciel

Comme la reconfiguration et la redistribution ne ciblent pas le composant à son échelle, mais à l'échelle du système, leurs effets ne sont pas représentés dans la figure 2.2, même si elles peuvent impacter indirectement l'état. Un composant peut ne pas être affecté par une reconfiguration ou une redistribution, ainsi son état reste inchangé. Par ailleurs, la reconfiguration peut mener à une adaptation à l'échelle du composant, et aussi à un ajout ou une suppression d'un composant : ajouter un composant consiste à l'activer (après l'avoir installé si nécessaire) et le supprimer consiste à le désinstaller (après l'avoir désactivé si nécessaire). La redistribution peut mener au déplacement d'un composant d'un site consommateur à un autre ; ainsi, elle consiste en une désactivation (et possiblement une désinstallation) sur le site d'origine d'une part, (une possible installation et) une activation sur le site destinataire d'autre part.

2.2.5 Conception

Les activités introduites précédemment sont liées à un point de vue opérationnel sur le déploiement. En complément, il faut souligner que la conception est une activité essentielle du déploiement.

— **Conception.** La conception à pour but de produire un plan de déploiement.

Le plan de déploiement peut être construit « à la main » ou calculé plus ou moins automatiquement. Le plan de déploiement peut être combiné avec une planification temporelle de son application.

2.2.6 Chronologie

La figure 2.3 présente la chronologie du déploiement en relation avec la conception et l'exécution du logiciel déployé. Les activités de déploiement se déroulent avant, pendant et après l'exécution du logiciel. Avant l'exécution, le logiciel est installé et activé : cette phase est communément appelée **déploiement initial**. Pendant l'exécution, le plan de déploiement est en vigueur et il peut être modifié par des opérations de déploiement. Après l'exécution,

le logiciel est désactivé et peut être désinstallé. Le temps de réalisation du déploiement recouvre donc le temps d'exécution du logiciel.

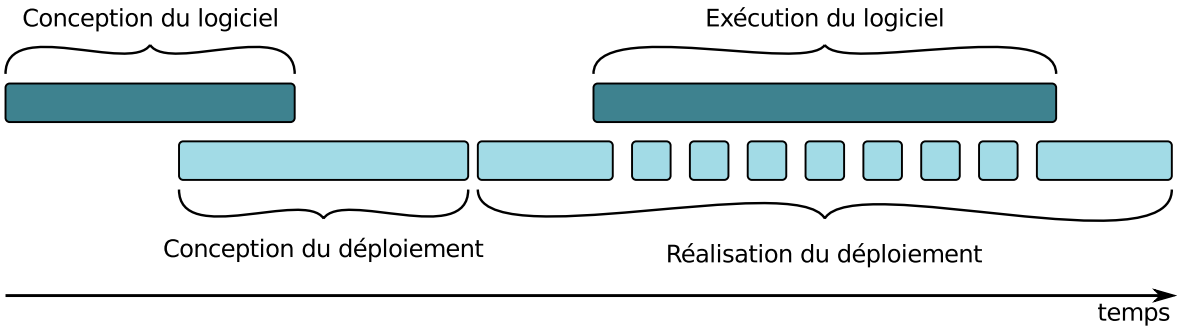


Figure 2.3 – Chronologie du déploiement

Le déploiement n'est donc pas une opération limitée dans le temps, en particulier dans le cadre de systèmes instables, à cause de la durée de vie des applications et des évolutions du logiciel et du domaine de déploiement (apparition et disparition d'appareils, perte de connexion, etc.). Nous définissons donc deux formes particulières de déploiement : le déploiement incrémental et le déploiement continu.

— **Déploiement incrémental**. Le déploiement incrémental consiste à déployer un nouveau composant au sein d'un système logiciel déjà déployé.
— **Déploiement continu**. Le déploiement continu consiste à effectuer une ou des opérations de déploiement sur un appareil entrant ou sortant du domaine.

Le déploiement incrémental concerne donc les opérations liées à la dynamique du système logiciel alors que le déploiement continu concerne celles qui sont liées à la dynamique du domaine de déploiement. Les deux activités impactent le plan de déploiement.

2.3 Analyse du problème

Dans le cadre du projet INCOME, nous avons contribué à la définition d'un scénario de référence appelé 4ME pour « Mobilité MultiModale Multi-Échelle » (cf. section 1.3) [Arcangeli et al., 2013]. Ce scénario définit un système de guidage porte-à-porte multimodal pour les usagers véhiculés ou utilisant les transports en commun d'une ville. Le système de guidage propose des services d'information à destination des voyageurs se déplaçant avec des moyens de transports multimodaux (pédestre, vélo en libre service, taxis, réseau de transports en commun) et un accès ubiquitaire et en temps réel aux informations (horaires de passages des transports en commun, cartes d'itinéraires, informations de réseaux sociaux de passagers). 4ME est donc une application consommatrice d'informations de contexte et, par conséquent, s'appuie sur une autre application, le gestionnaire de contexte multi-échelle, qui collecte, construit (calcule, infère), achemine (distribue, diffuse) et présente une information de contexte de qualité [Arcangeli et al., 2012a].

Dans cette thèse, nous nous intéressons au déploiement de ce type d'applications réparties multi-échelles (4ME et le gestionnaire de contexte). L'objectif est de proposer une solution pour le déploiement de systèmes répartis multi-échelles. Par le terme « solution », nous désignons un ensemble d'éléments susceptibles de constituer notre proposition tels une méthode, un processus, des outils, du logiciel, etc.

2.3.1 Cadre d'analyse

Afin d'identifier les problèmes posés par le déploiement de systèmes logiciels répartis multi-échelles et les exigences qu'une solution de déploiement doit satisfaire, nous formulons quelques questions qui vont guider notre analyse du problème et, dans le chapitre suivant (cf. section 3.2), notre étude de l'état de l'art :

— « *Qu'est ce qui est déployé ?* » La première question concerne la nature du système logiciel à déployer et des unités de déploiement, leur hétérogénéité et leur nombre, la dynamique du système et son ouverture.
— « *Où le logiciel est-il déployé ?* » De manière symétrique, il faut étudier la nature du domaine de déploiement ciblé, sa dynamique et son ouverture, le nombre d'appareils et leur hétérogénéité, le type de réseau qui le supporte, ainsi que les propriétés d'administration.
— « *Comment le déploiement est-il effectué ?* » Cette question concerne la manière de déployer du point de vue de la conception et du point de vue de la réalisation (expression du plan, mise en œuvre, activités prises en compte, etc.).

2.3.2 Analyse

Dans cette partie, nous analysons le problème conformément au cadre défini ci-dessus et nous formulons les exigences générales auxquelles un système de déploiement doit répondre, ainsi que quelques hypothèses. Certaines de ces exigences seront reprises dans les chapitres suivants et raffinées en exigences plus spécifiques. Ces exigences sont listées dans l'annexe C.

2.3.2.1 L'application

L'application à déployer est une application répartie à base de composants logiciels. La composition de l'application implique un déploiement unitaire de chacun des composants et une configuration des liens entre composants afin de les assembler en une application fonctionnelle. Le système de déploiement doit pouvoir déployer une application sans qu'aucun composant n'aie déjà été déployé.

Les composants de l'application sont de types divers. Par exemple, au sein d'un gestionnaire de contexte réparti multi-échelle certains composants collectent des données, d'autres infèrent des informations, les acheminent, ou les présentent, etc. Le système de déploiement doit prendre en compte cette diversité de types, y compris les différentes versions possibles d'un composant.

De manière générale, on peut souhaiter assembler et administrer des composants issus de modèles de composants différents. Dans le cadre de ce travail, nous nous limiterons à un unique modèle de composant (hypothèse d'homogénéité).

Pour ce qui est du nombre, on peut considérer que le gestionnaire de contexte réparti multi-échelle et l'application 4ME sont constitués de plusieurs milliers de composants répartis sur le domaine. Le système de déploiement doit donc supporter un nombre de composants de l'ordre de plusieurs milliers.

Ces applications ont par ailleurs une durée de vie suffisamment longue pour être en situation d'évoluer dynamiquement par des ajouts ou des retraits de composants, des mises

à jour, etc. Le système de déploiement doit donc supporter la dynamique de l'application et opérer alors que l'application est en cours d'exécution et sans l'interrompre.

2.3.2.2 Le domaine de déploiement

Le domaine de déploiement est un ensemble d'appareils hétérogènes : réseaux de capteurs, équipements personnels des utilisateurs (montres connectées, smartphones, tablettes, ordinateurs personnels, etc.), serveurs de proximité (par ex. dans les bus ou aux arrêts de bus pour collecter et agréger des données), serveurs distants, machines au sein d'un Cloud (pour calculer et inférer de l'information), etc. Notre solution doit prendre en compte cette hétérogénéité matérielle, ainsi que celle des réseaux qui interconnectent ces appareils, et permettre le déploiement sur différents types d'appareils.

Néanmoins, pour les réseaux de capteurs d'une part, et les machines du Cloud d'autre part, les problèmes du déploiement sont bien particuliers (par ex. celui de la consommation énergétique pour les capteurs) et il existe des solutions spécifiques et adaptées sur lesquelles il est possible de s'appuyer [Marrón et al., 2006, Hughes et al., 2012, Horré et al., 2011]. Comme, en pratique, ces appareils sont connectés au domaine par l'intermédiaire d'une machine passerelle, il est possible de considérer que les passerelles sont les limites du déploiement, et que le système de déploiement n'a pas à prendre en compte directement les capteurs et les machines du Cloud.

Nous considérerons de la même manière n'importe quel appareil à capacité trop limitée pour laquelle nous supposerons l'existence d'une machine passerelle (par ex. dans le domaine de la domotique). Les machines du Cloud, les capteurs et les appareils à capacité trop limitée ne font donc pas partie du domaine de déploiement. On peut cependant imaginer que, dans l'avenir, notre solution pourra opérer sur certains réseaux de capteurs si ceux-ci évoluent et sont dotés de ressources matérielles et logicielles suffisantes.

Comme précisé précédemment, le système doit supporter le déploiement sur des domaines composés de milliers d'appareils.

Ces appareils sont autonomes et fonctionnent indépendamment les uns des autres. Une contrainte, liée à cette autonomie, réside dans la possibilité pour un appareil d'entrer dans le domaine alors que l'application est en cours d'exécution, ou inversement d'en sortir. Ces événements peuvent résulter de la mobilité, de la mise en marche ou de l'arrêt d'un appareil. Par exemple, un utilisateur de 4ME peut décider de mettre en marche son smartphone ou de lancer l'application, ou se déplacer physiquement et apparaître (entrer dans le réseau). Le domaine de déploiement est donc ouvert et évolue dynamiquement. Les possibles variations en matière de disponibilité et de qualité des ressources et des services utiles à l'application déployée sont autre source de dynamique. Le système de déploiement doit prendre en compte cette dynamique du domaine de déploiement et adapter le plan de déploiement aux changements.

Une autre contrainte provient du mode d'administration du domaine et des droits en matière de déploiement. Dans le contexte envisagé, chaque appareil (par exemple, les smartphones) ou groupe d'appareils a son propre propriétaire et son administrateur qui détient le droit d'effectuer des opérations de déploiement. Vu le nombre, on ne peut pas supposer un administrateur unique. Afin que le système de déploiement puisse opérer sur les appareils du domaine malgré l'administration multiple, nous ferons l'hypothèse simplificatrice d'une cession (plus ou moins automatique) des droits par l'administrateur. Autrement dit, nous ne

traitons pas ce problème et nous supposons qu'en acceptant notre solution, l'administrateur accepte aussi de céder une partie de ses droits.

2.3.2.3 La conception et la réalisation du déploiement

Le déploiement d'un système suppose la définition, ou conception, d'un plan de déploiement puis la réalisation du déploiement conformément à ce plan.

La dynamique et l'instabilité du domaine de déploiement posent un problème majeur. En effet, il n'est pas possible de connaître *a priori* l'ensemble des appareils qui constituent le domaine à un instant donné. Il est donc impossible pour un concepteur d'exprimer directement un plan de déploiement dans lequel les appareils sont désignés explicitement. Du fait du nombre (d'appareils, mais aussi de composants logiciels), il n'est pas non plus souhaitable que le concepteur doive exprimer un plan de manière complète.

Ceci a deux conséquences. D'une part, puisqu'il n'est ni souhaitable ni possible d'exprimer un plan, le système de déploiement doit permettre l'expression des propriétés attendues en matière de déploiement, c'est-à-dire de spécifier le plan de déploiement. Pour cela, le concepteur doit bénéficier d'abstractions, d'autant qu'il n'est pas forcément expert en déploiement mais plutôt de l'application déployée (dans le projet INCOME, on fait l'hypothèse d'utilisateurs « grand public » qui peuvent être parties prenantes dans le processus de déploiement). En particulier, parmi ces abstractions, le concepteur doit disposer de moyens pour exprimer des propriétés portant sur une partie du domaine en fonction de l'échelle : par exemple, il doit pouvoir spécifier le déploiement d'un composant à l'échelle d'une ville ou d'un quartier, ou sur les appareils d'un certain type (par exemple, tous les smartphones) ou connectés à un réseau donné, etc. Pour cela, un système de déploiement doit offrir un langage dédié (*Domain Specific Language* ou DSL) au déploiement de systèmes logiciels répartis multi-échelles.

D'autre part, cette spécification du déploiement doit être traduite en un plan de déploiement en fonction de l'état du domaine, que ce soit pour le déploiement initial ou en cours d'exécution de l'application déployée. Le système de déploiement doit donc permettre de générer un plan de déploiement initial à partir de la spécification et de l'état du domaine au moment du lancement de l'application. Il doit ensuite adapter dynamiquement le plan de déploiement en fonction de l'état courant du domaine de sorte que les spécifications demeurent respectées.

Par conséquent, pour être « sensible au contexte », le système de déploiement doit pouvoir observer, construire et manipuler un état significatif du domaine de déploiement.

Ces différentes opérations doivent être organisées et automatisées dans le cadre d'un processus. Du fait du nombre de composants et d'appareils, les opérations de déploiement initial (installation, activation) ne peuvent pas être laissées à la seule initiative des administrateurs de chacun des appareils (c'est-à-dire en mode *pull*). Au contraire, le mode *push*, dans lequel le déploiement d'une application est initié et commandé à distance de manière centralisée, doit être privilégié. En revanche, toujours pour des raisons de nombre, les autres activités doivent être gérées de manière décentralisée et localement, sans se référer (ou le moins possible) à un organe centralisé. Alors, pour ne pas faire localement appel à un opérateur humain, le système doit mettre en œuvre un « système de déploiement » qui joue le rôle d'opérateur.

2.3.3 Synthèse

Cette analyse met en évidence certaines exigences pour le déploiement des systèmes logiciels répartis multi-échelles. Dans un premier temps, un concepteur doit pouvoir exprimer une spécification du plan de déploiement sans avoir à exprimer directement un plan. Au contraire, ce plan doit être généré à partir de cette spécification, et doit également être fonction de l'état initial du domaine. Ensuite, le plan doit pouvoir évoluer dynamiquement en fonction de la dynamique de l'application et de celle du domaine. Ces évolutions doivent être prises en compte de manière autonome et décentralisée, au niveau middleware. Enfin, toutes ces activités doivent être organisées et coordonnées dans le cadre d'un processus automatisé.

Contribution

Le déploiement de logiciel est un processus qui organise et orchestre un ensemble d'activités ayant pour but de rendre le logiciel disponible à l'utilisation et de le maintenir à jour et opérationnel. Une phase de conception, qui a pour objectif la production d'un plan de déploiement, précède la phase de réalisation. Après réalisation du déploiement initial, le déploiement incrémental et le déploiement continu adaptent le système logiciel en cours d'exécution tout au long de sa vie. Dans le cas d'un système réparti multi-échelle, le domaine de déploiement est ouvert et n'est que partiellement connu en phase de conception. Il n'est donc pas possible (ni souhaitable) d'exprimer directement le plan de déploiement. Au contraire, le concepteur doit le spécifier sous la forme de propriétés à respecter. Le plan doit ensuite être généré puis réalisé automatiquement en fonction du contexte d'exécution. Il doit évoluer dynamiquement de manière autonome et décentralisée.

3 État de l'art sur l'automatisation du déploiement

Problématique

Du fait de la distribution et de l'ubiquité, du nombre et de l'hétérogénéité, de la mobilité et de l'évolutivité des systèmes logiciels, et plus généralement de la dynamique et de la complexité des structures matérielles et logicielles, le déploiement demande de l'automatisation et de l'autonomie. Différentes solutions ont été proposées. Leur capacité à répondre aux exigences du déploiement des systèmes répartis multi-échelles doit être analysée.

3.1 Introduction

Traditionnellement, le déploiement de logiciel est géré « à la main » par un humain qui explicite sur quel(s) appareil(s) le ou les composants doivent être installés, activés, etc. Aujourd'hui, du fait de la distribution, de la mobilité et de l'ubiquité, du nombre d'appareils et de leur hétérogénéité, de la complexité et de l'évolutivité des systèmes logiciels, le déploiement exige plus d'automatisation. Il requiert donc des méthodes et des outils appropriés pour la conception (par ex. pour exprimer des contraintes et des exigences de déploiement), le contrôle et l'automatisation.

L'automatisation du déploiement n'est pas un problème nouveau. Différentes technologies apportent des solutions telles que Redhat Package Manager [Bailey, 2000], Microsoft Windows, Microsoft .Net, Entreprise JavaBeans [Ora2], Corba Component Model [Obj2], OSGi [OSGi Alliance, 2009] ou encore les produits de VMware [VMware Inc., 2008] pour la virtualisation. Cependant, ces solutions techniques de déploiement sont souvent limitées au mode *pull*, c'est-à-dire à l'installation à la demande d'un client, et à l'installation. Les étudier n'entre pas dans le cadre de cet état de l'art, toutefois ces technologies sont présentées et comparées par exemple dans [Dearle, 2007] ou [Heydarnoori, 2008].

Dans ce chapitre, nous dressons un état de l'art des travaux de recherches sur le déploiement automatique [Arcangeli et al., 2015]. Nous commençons par proposer un cadre d'analyse des différents travaux en reprenant et complétant celui présenté à la section 2.3.1. Puis nous présentons et analysons dix-sept travaux récents, et une synthèse est proposée en

référence à ce cadre d'analyse. L'étude de ces travaux montre que dans un contexte de distribution, d'hétérogénéité, de nombre, de dynamique et d'ouverture, les solutions existantes sont incomplètes et potentiellement inefficaces ou inutilisables. De plus, le niveau d'abstraction et d'expressivité de la conception s'avère limité au vu de la complexité du problème du déploiement.

Ce chapitre est organisé comme suit. La section 3.2 présente le cadre d'analyse. Ensuite, la section 3.3 étudie les travaux de recherche selon le cadre analytique et la section 3.4 en fournit une synthèse.

3.2 Cadre d'analyse et critères

Nous avons défini à la section 2.3.1 un cadre d'analyse pour l'étude du déploiement de systèmes répartis multi-échelles. Nous reprenons ce cadre comme grille de lecture des travaux de l'état de l'art. Cependant, nous raffinons la dernière question qui porte sur la manière dont le déploiement est effectué, en deux questions, l'une portant sur la conception du déploiement, et l'autre sur sa réalisation :

— « *Comment le déploiement est-il conçu ?* » Cette question concerne ce qui doit être exprimé (qui peut aller d'un plan de déploiement – les commandes de déploiement qui doivent être réalisées – jusqu'à une spécification de propriétés liées au déploiement qui devra être interprétée) et le niveau d'expressivité. Nous étudions aussi quel acteur exprime ou spécifie ce plan et comment les différentes parties prenantes sont impliquées. Comme nous nous intéressons à l'utilisabilité des solutions de déploiement, nous portons une attention particulière au niveau d'expertise attendu des parties prenantes.

— « *Comment le déploiement est-il réalisé ?* » Cette question est liée aux activités de déploiement, et à la manière dont elles sont réalisées, à leurs contraintes et à leurs hypothèses. Nous considérons également l'architecture du système de déploiement, qui peut être centralisée ou décentralisée. Nous étudions enfin si et comment les solutions utilisent un programme d'amorce (bootstrap) qui supporte le déploiement sur un appareil et doit être opérationnel avant la réalisation du déploiement.

Notre analyse se base sur sept critères principaux : **l'unité de déploiement**, **le domaine de déploiement**, **l'expression des propriétés**, **l'expertise du concepteur du déploiement**, **les activités de déploiement**, **le contrôle du déploiement** et **la nature du bootstrap**. La figure 3.1 met en évidence les liens entre ces points et les questions proposées.

Dans la littérature, il est possible de trouver d'autres cadres d'analyse. Par exemple, dans [Heydarnoori, 2008], Heydarnoori compare plusieurs techniques de déploiement en se focalisant sur les activités supportées et en classant ces techniques en huit catégories – dirigée par la qualité de service (*QoS-driven*), dirigée par les modèles (*model-driven*), à base d'agent, orientée grille, etc. – mais sans justifier la pertinence de son cadre d'analyse.

3.3 Étude des travaux

Dans cette section, nous passons en revue l'état de l'art du déploiement automatique (travaux de recherche et projets associés). Afin d'organiser cette étude – principalement pour faciliter la lecture de l'état de l'art, nous divisons ces travaux en cinq catégories en fonction

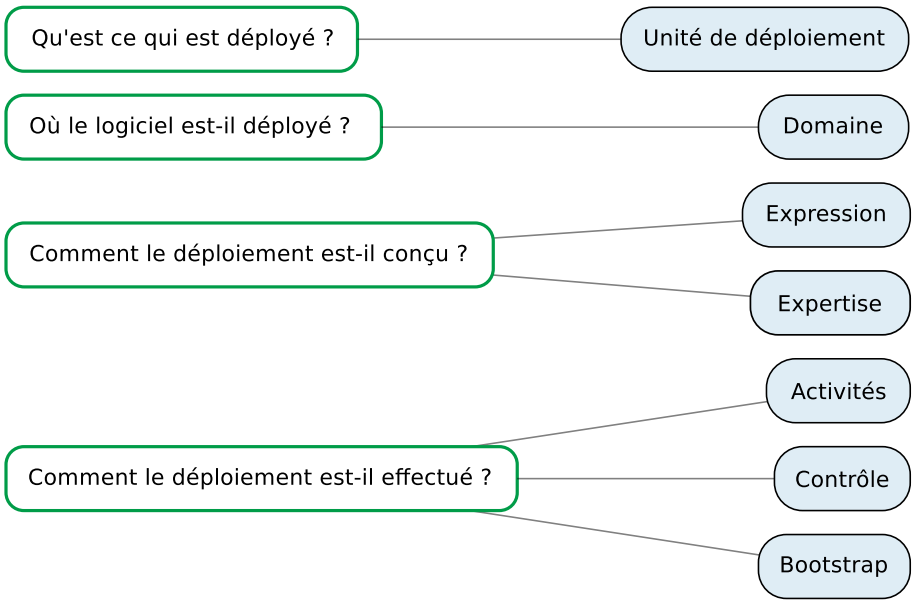


Figure 3.1 – Cadre d'analyse et critères

de leurs objectifs :

1. étendre OSGi ou utiliser les facilités que fournit OSGi,
2. relever l'humain des tâches de déploiement,
3. allouer des ressources dynamiquement pour le calcul,
4. gérer des appareils mobiles et disposant de ressources limitées,
5. fournir une abstraction afin de faciliter la conception.

Pour chaque travail, nous proposons une présentation du *problème*, une description de la *solution proposée* et une *discussion* en relation avec les question et les points principaux soulevés dans la section 3.2. Enfin, un résumé des résultats organisé en fonction du cadre analytique est donné dans la section 3.4.

3.3.1 Automatisation basée sur OSGi

OSGi [OSGi Alliance, 2009] est une spécification qui définit différentes fonctionnalités pour le déploiement et l'administration distante d'applications à base de composants. Le framework OSGi permet de gérer toutes les activités de déploiement de composants appelés *bundles*, qui sont les unités de déploiement). Un bundle est un fichier JAR avec un composant Java contenant un manifeste (métadonnées textuelles décrivant par exemple les dépendances et les exigences de déploiement et une combinaison de fichiers bytecode Java, du code natif et des ressources associées. OSGi permet l'installation, la désinstallation, l'activation, la désactivation et la mise à jour de composants à l'exécution sans avoir besoin de redémarrer l'ensemble du système. Les implémentations d'OSGi (commerciales et open source) existent pour différents appareils hétérogènes, allant du smartphone (au moins Android 1.5 ou Symbian S60) aux ordinateurs personnels, en passant par les tablettes, les PCs ultra-mobiles, les PCs de voitures et les ordinateurs portables. OSGi masque l'hétérogénéité des hôtes et les aspects techniques bas-niveau du déploiement. Cependant, cette solution est restreinte à un seul hôte, le déploiement est local.

OSGi a été étendu pour répondre au besoin de déploiement de composants distribués. Par exemple, dans le projet UseNet [USE], les auteurs s'intéressent à des scénarios innovants et à du *Machine-to-Machine* (M2M, la communication inter-machines) expérimental tout en visant à permettre l'usage ubiquitaire des services M2M. Ils utilisent la technologie OSGi pour l'exécution du déploiement sur différents types d'appareils connectés à différents réseaux de communication, mais sans traiter la dynamique de la topologie des hôtes. Dans cet état de l'art, nous étudions des travaux qui se basent sur OSGi et qui visent à l'étendre dans un but précis, comme le déploiement de composants Fractal [Desertot et al., 2006] ou la gestion distribuée de modules [Rellermeyer et al., 2007].

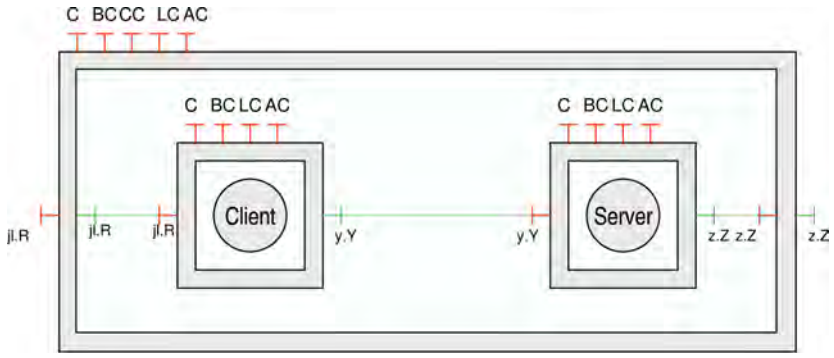
3.3.1.1 FROGi

Problème Le modèle hiérarchique de composant Fractal présente plusieurs limitations quant au déploiement : en particulier le concept d'unité de déploiement est différent de celui des spécifications de Fractal, et la dynamique du déploiement est très restrictive. Fractal peut être enrichi avec des capacités de déploiement efficaces.

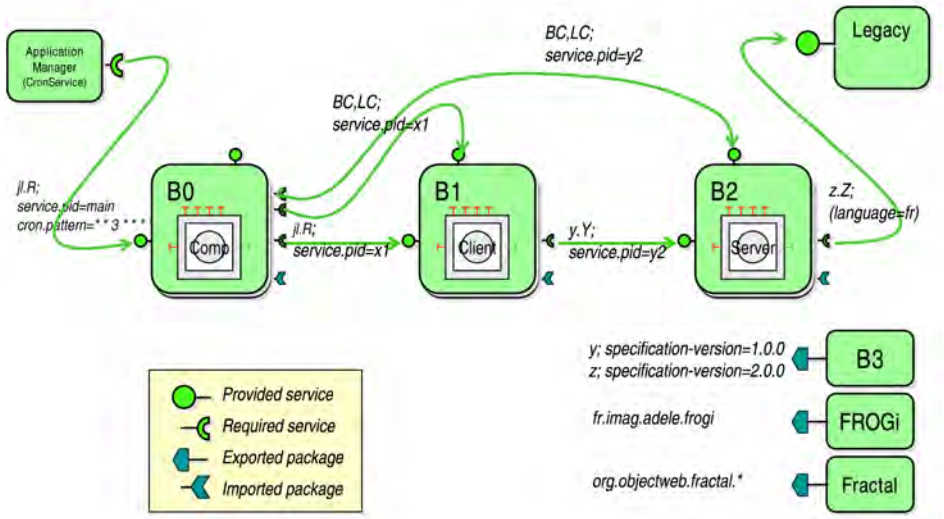
Solution Desertot *et al.* suggèrent de combiner le modèle de composants Fractal avec OSGi [Desertot et al., 2006]. D'un côté, FROGi améliore Fractal en facilitant le déploiement de composants Fractal (originels ou composites) en utilisant la plateforme de services d'OSGi. D'un autre côté, il fournit aux développeurs OSGi la possibilité d'utiliser des composants basés sur Fractal.

Dans FROGi, une application à base de composants Fractal est empaquetée en un ou plusieurs bundles et la plateforme OSGi rend les composants disponibles en tant que services. La figure 3.2 montre une application Fractal empaquetée et déployée en utilisant OSGi. La figure 3.2a montre le composant composite créé à partir des deux composants originels, Client et Serveur. La figure 3.2b montre les trois composants (Comp étant le composant composite), chacun d'entre eux empaqueté dans un bundle (B0, B1, B2). Leurs interfaces fournies et requises sont considérées comme des services de bundles. Pour l'empaquetage des composants, FROGi étend l'ADL (*Architecture Description Language*, langage de description d'architecture) de Fractal afin de permettre la spécification de bundles, leurs versions et propriétés. C'est la plateforme OSGi qui s'occupe du déploiement. Pendant l'installation d'un bundle, OSGi résout automatiquement les dépendances de code. Puis, les bundles sont activés et un bootstrap crée des instances de composants. À l'exécution, FROGi supporte la gestion du cycle de vie des composants ainsi que leur liens et la reconfiguration dynamique.

Discussion Cette solution est basée sur une implémentation libre d'OSGi, et la plateforme Julia, qui est une implémentation en Java du framework Fractal. Elle supporte le déploiement local de composants Fractal mais pas leur distribution : un système de composants, intégré en un ou plusieurs bundles, peut être déployé mais localement seulement (il n'y a pas de plan de déploiement). Les activités couvertes sont l'installation, la désinstallation, l'activation, la désactivation et la mise à jour, tandis que le niveau de dynamique est celui d'OSGi. L'appareil visé doit contenir une plateforme OSGi et le bootstrap de FROGi. Le composant et l'appareil doivent être explicitement indiqués, et le concepteur du déploiement doit avoir une expertise de l'ADL de Fractal et doit être capable de construire des bundles OSGi.



(a)



(b)

Figure 3.2 – Une application Fractal (a) et son empaquetage en bundles OSGi (b) [Desertot et al., 2006]

3.3.1.2 R-OSGi

Problème OSGi est conçu pour permettre le déploiement et l'exécution d'une application locale. La difficulté est de permettre à une application OSGi d'être répartie sans perdre les propriétés du modèle OSGi.

Solution R-OSGi (*Remoting-OSGi*) est une plateforme middleware distribuée qui étend les spécifications d'OSGi pour supporter la gestion répartie des modules sans interruption et efficacement [Rellermeyer et al., 2007]. Elle permet à une application OSGi centralisée d'être, automatiquement et d'une manière transparente, répartie et de fonctionner en utilisant des mandataires.

R-OSGi se base sur la génération dynamique de mandataires et l'injection de types pour assurer la consistance du typage. Les mandataires fournissent des services OSGi localement, et cachent l'invocation de services à travers le réseau. La seule différence par rapport aux services OSGi standards est que ces services doivent prendre en compte la distribution afin d'effectuer des opérations spécialisées, par exemple pour la gestion du système. La découverte de services est réactive et efficace. Les services sont enregistrés et localisés au

moyen d'un registre distribué implémenté avec le *Service Location Protocol* (protocole de localisation de service), qui complète le registre de service centralisé d'OSGi. À l'exécution, les techniques développées pour la gestion de module centralisée d'OSGi, comme les chargement dynamique, déchargement et lien des modules, sont utilisées pour gérer la dynamique du domaine de déploiement (par ex. pannes partielles).

Discussion R-OSGi facilite le déploiement de bundles OSGi, de telle manière qu'ils peuvent interagir à distance à l'exécution à travers un réseau d'appareils qui hébergent une plateforme OSGi.

R-OSGi hérite des propriétés et des capacités d'OSGi. Ici, le concepteur du déploiement doit établir un plan de déploiement explicite et doit être expert dans la technologie OSGi.

3.3.2 Remplacement de l'humain dans les tâches de déploiement

Plusieurs travaux observent que le déploiement implique un ensemble de décisions et d'actions de bas-niveau complexes pour un opérateur humain. Par conséquent, ils ont pour objectif de réduire cette complexité en renforçant l'autonomie du système de déploiement. C'est le cas pour Software Dock [Hall et al., 1999] et QUIET [Manzoor and Nefti, 2010] qui utilisent des agents mobiles pour déployer automatiquement à travers le réseau. Disnix propose une autre solution pour le déploiement automatique basée sur des modèles [van der Burg and Dolstra, 2014]. Enfin, RAC automatise entièrement l'installation et la configuration de machines virtuelles (VM, *virtual machines*) pour l'informatique dans le Cloud, et enlève l'humain de la boucle [Liu, 2011].

Également, le projet Selfware [SEL] vise à limiter l'implication de l'humain dans l'administration du système afin de réduire les erreurs et de permettre des réactions automatiques à certaines situations d'exécution spéciales. Selfware propose une plateforme logicielle qui permet aux systèmes distribués d'être construits avec une administration autonomique (auto-réparation, etc.). Ainsi, des applications JEE existantes sont encapsulées dans des composants Fractal [OW2] qui peuvent être dynamiquement reconfigurés afin de pouvoir gérer des problèmes de pannes ou des problèmes de passage à l'échelle. Cependant, cette solution est limitée aux applications JEE déployées sur des appareils connus à l'avance.

3.3.2.1 Software Dock

Problème À la fin des années 90, avec l'émergence de l'Internet, le processus d'installation manuelle de logiciels en utilisant un support physique comme un CD-ROM a été abandonné. Depuis cette époque, la connectivité réseau permet aux producteurs de logiciels d'offrir des services de déploiement distants de haut-niveau aux consommateurs. Ainsi, Hall *et al.* proposent Software Dock, un framework de déploiement distribué à base d'agents qui permet la coopération entre les producteurs et les utilisateurs du logiciel [Hall et al., 1999].

Solution La *Deployable Software Description* (DSD, description du déploiement du logiciel) est un élément principal de cette solution. C'est un langage standardisé utilisé pour décrire un système logiciel à travers un ensemble de propriétés sémantiques liées aux caractéristiques et aux contraintes côté consommateurs d'une part, et à la configuration de ces éléments d'autre part.

L'architecture de Software Dock est distribuée et basée sur deux types d'élément : le *release dock* sur le site producteur, et le *field dock* sur le site consommateur qui fournit des informations locales à propos des ressources et de la configuration, des logiciels déjà déployés, etc. Le déploiement se base sur des agents mobiles qui effectuent des activités de déploiement. Les agents se déplacent d'un *release dock* vers un *field dock* en transportant le logiciel mis à disposition et la description DSD (la mobilité des agents est cependant limitée à un saut sur un site décidé d'avance). Sur le site consommateur, ils interagissent avec le *field dock* et effectuent une configuration spécifique et le déploient en interprétant localement la description DSD (les agents peuvent être plus ou moins spécialisés pour une activité de déploiement). Un service d'évènements (*event service*) distribué se charge de la connectivité entre les producteurs et les consommateurs (cf. figure 3.3). Se basant sur un mécanisme de *publish/subscribe*, le *release dock* peut générer des évènements, par exemple dans le cas d'une mise à jour. Ces évènements sont capturés par les agents souscripteurs sur le site consommateur, ce qui peut engendrer une mise à jour ou une reconfiguration.

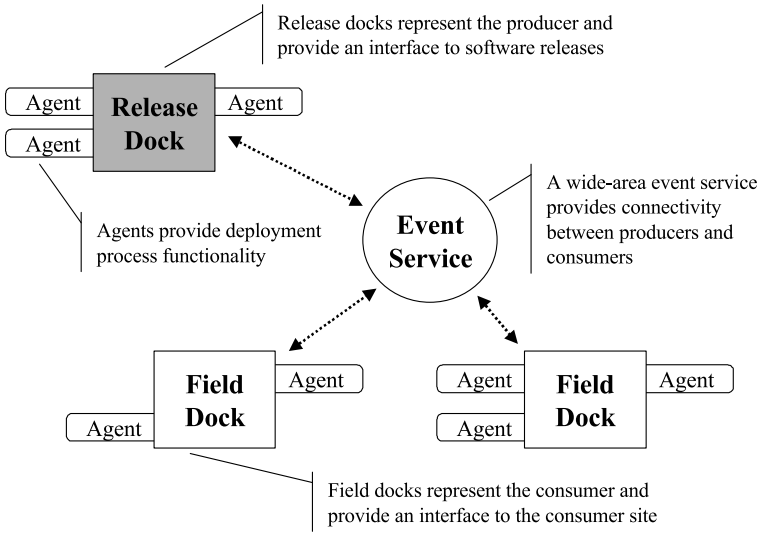


Figure 3.3 – Architecture de Software Dock [Hall et al., 1999]

Discussion Le processus de déploiement lui-même est distribué à travers le réseau mais les applications sont déployées localement en tant qu'unités de déploiement. Le déploiement est spécifique au site consommateur grâce à l'interprétation locale de descripteurs, et la prise en compte de la dynamique du domaine de déploiement (apparition d'appareils) s'effectue avec le service d'évènement. L'implémentation utilise la plateforme d'agents mobiles Voyager qui permet de s'abstraire des problèmes d'hétérogénéité du domaine de déploiement. De plus, les appareils doivent héberger le *field dock*, qui est le bootstrap. Plusieurs activités sont prises en compte : le dépôt et le retrait de l'application sur le site producteur, l'installation et la désinstallation, la mise à jour, l'adaptation et la reconfiguration sur le site consommateur. Dans ce contexte, le concepteur du déploiement doit avoir une forte expertise dans le déploiement. Il doit établir la configuration et exprimer les contraintes et les dépendances dans le format DSD.

3.3.2.2 QUIET

Problème Les assistants d'installation traditionnels permettent à l'utilisateur de participer et d'exprimer ses préférences au travers d'une interface graphique. Manzoor et Nefti ont essayé de limiter cette interaction lorsque elle n'est pas nécessaire, en favorisant l'automatisation de l'installation du logiciel sur un réseau d'ordinateurs personnels.

Solution QUIET est un framework pour l'installation et la désinstallation automatique à travers le réseau [Manzoor and Nefti, 2010], qui supporte l' « installation silencieuse et sans surveillance » (*silent and unattended installation*), c'est-à-dire une installation sans interaction avec l'utilisateur. L'installation silencieuse utilise le gestionnaire de paquet SUIPM (*Silent Unattended Installation Package Manager*, gestionnaire de paquet silencieux et sans surveillance) [Manzoor and Nefti, 2008]. Au niveau réseau, un système multi-agent gère les ressources (bande passante, mémoire, disque, etc.) du domaine de déploiement et déploie intelligemment et efficacement les SUIPM sur les sites consommateurs en fonction des conditions réseau. Les agents supportent l'autonomie et la décentralisation, leur mobilité permet une meilleure couverture du réseau en cas de pannes.

Le réseau est divisé en sous-réseaux, chacun d'eux correspondant à un ensemble de sites consommateurs, gérés par des sous-serveurs. La figure 3.4 décrit comment le système multi-agent opère et les comportements des différents types d'agents : chacun d'eux est en charge d'une opération spécifique et peut créer des agents afin d'exécuter des tâches annexes. Premièrement, un *Master Controller Agent* (MCA, agents de contrôle maîtres) charge l'application à installer et la description du domaine de déploiement à partir d'un fichier XML, et crée des *File Transfer Agents* (FTA, agents de transfert de fichiers) pour effectuer le transfert des fichiers de configurations sur les sous-serveurs. Un MCA crée aussi des *Controller Agents* (CA, agents de contrôle) qui migrent sur les sous-serveurs et sont responsables de l'installation sur les sous-réseaux. Dans un sous réseau, un CA crée des FTAs pour effectuer le transfert des fichiers de configurations sur le site consommateur et des *Installer Agents* (IA, agents d'installation) et des *Verification Installer Agents* (VIA, agents vérificateurs d'installation) qui migrent sur le site consommateur. Le CA est aussi responsable de l'envoi de données de contexte observées concernant le réseau et le processus au MCA, ce qui l'aide à construire une base de connaissance qui va lui permettre de prendre des décisions intelligentes. Éventuellement, sur le site consommateur, un IA installe le paquetage SUIPM et un VIA vérifie que l'installation s'est correctement effectuée en accord avec les fichiers de configuration.

Ensuite SUIPM peut installer l'application localement. La désinstallation s'effectue selon les mêmes principes que l'installation.

Un système d'enregistrement est utilisé pour la récupération du système en cas d'erreur. Tous les agents sont responsables de la supervision de l'agent qu'ils créent. L'agent créateur présume que l'agent créé est mort quand il ne reçoit plus de message d'enregistrement ; puis il crée un autre agent du même type qui reprend le processus de déploiement au même endroit, grâce aux enregistrements.

Discussion Le framework QUIET supporte le déploiement distribué, décentralisé et en parallèle d'un unique composant (SUIPM et puis une application MS Windows) à travers un réseau d'ordinateurs personnels. Son implémentation utilise la plateforme d'agents distribués Jade qui est supposée installée sur les appareils du domaine. Les utilisateurs

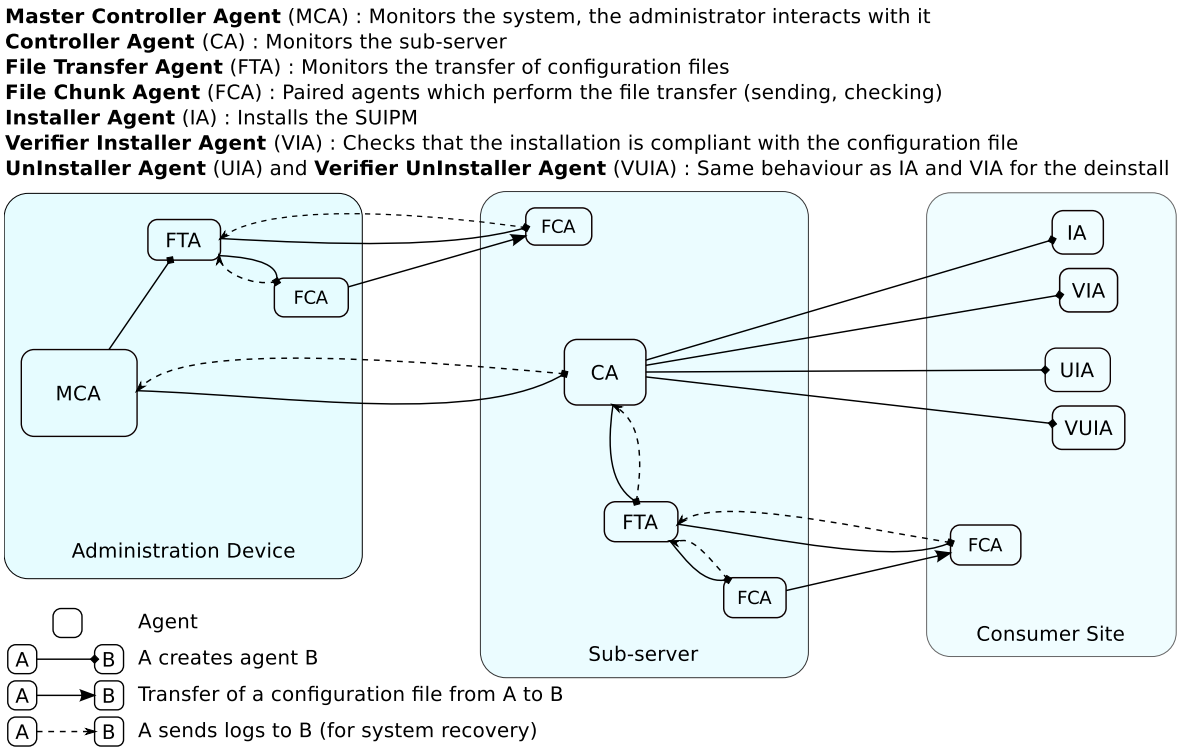


Figure 3.4 – Système multi-agent de QUIET

de QUIET peuvent lancer un dépôt de l'application (*via* SUIPM), une installation ou une désinstallation. Le concepteur du déploiement doit saisir les configurations du logiciel et de réseau (chemins, adresses, sous-serveurs, etc.) dans un fichier XML. Il doit donc avoir une certaine expertise dans le déploiement.

3.3.2.3 Disnix

Problème Plusieurs outils de déploiement sont spécifiques à un type de composant particulier (par exemple au déploiement d'Entreprise JavaBeans). Pourtant, les systèmes orientés service sont souvent composés de composants interdépendants, distribués et potentiellement hétérogènes qui fournissent les services. Le domaine de déploiement peut être hétérogène aussi, et changer continuellement à cause de pannes d'appareils ou de liens de communications. Cette instabilité nécessite un redéploiement dynamique. En conséquence, le déploiement de systèmes orientés service est complexe et gourmand en temps, et il est difficile de garantir des propriétés non fonctionnelles.

Solution Van de Burg *et al.* proposent Disnix, un outil pour le déploiement automatique et fiable de systèmes orientés service consistant en un ensemble varié de composants dans un domaine de déploiement hétérogène [van der Burg and Dolstra, 2010a], et une extension nommée DisnixOS qui supporte le déploiement d'infrastructure de composants [van der Burg and Dolstra, 2014]. Disnix se base sur Nix, un gestionnaire de paquets et un outil de déploiement local.

Le déploiement automatique distribué est basé sur des modèles déclaratifs. Le *service model* (modèle de service) définit les composants disponibles et distribuables, ainsi que leurs

propriétés et interdépendances. L'*infrastructure model* (modèle d'infrastructure) définit le domaine de déploiement. En dernier lieu, le *distribution model* spécifie la correspondance entre les services et le domaine, c'est-à-dire le plan de déploiement. Utiliser des modèles et des outils de transformation permet de dissimuler la complexité du déploiement aux concepteurs.

Un déploiement avec Disnix est lancé à partir d'un machine coordinatrice. Le processus consiste en la construction et l'installation de services, puis leur activation. En pratique, les codes sources sont compilés en utilisant un compilateur adéquat, les connecteurs (par ex. des pilotes de Java DataBase Connectivity) peuvent être générés, les composants sont installés de telle sorte que leurs dépendances (entre le composant et la machine hôte ou les autres composants) soient satisfaites, ensuite les services sont composés. La mise à jour est optimisée en la limitant au déploiement de nouveaux composants, à la désactivation de services obsolètes et à l'activation de nouveaux.

Dans [van der Burg and Dolstra, 2011], les auteurs proposent un framework de déploiement auto-adaptatif construit au-dessus de Disnix. Conjointement au service de découverte à l'exécution (pour les machines entrant dans le domaine de déploiement) et un générateur d'infrastructure, le framework génère une correspondance entre les composants et les machines en utilisant un *quality and service model* (modèle de qualité et de service) qui supporte l'expression d'une politique de distribution basée sur des attributs de qualité. La figure 3.5 illustre cette nouvelle architecture.

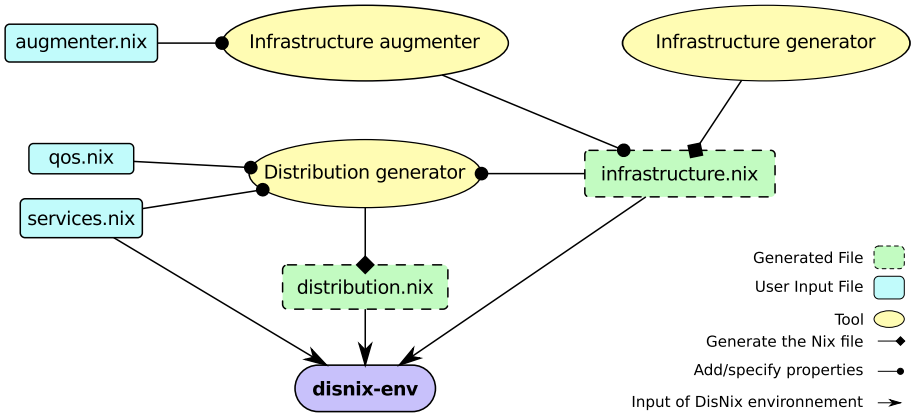


Figure 3.5 – Architecture étendue du déploiement de Disnix [van der Burg and Dolstra, 2011]

Discussion Disnix gère le déploiement de systèmes à base de composants hétérogènes sur les réseaux d'appareils hétérogènes. Les mises à jours de l'application sont prises en compte et la dynamique du domaine (apparition d'appareils et pannes d'appareils ou de liens réseau) est prise en compte dans la version étendue de Disnix qui supporte le redéploiement automatique. Les activités de déploiement supportées sont l'installation et la désinstallation, l'activation et la désactivation, la mise à jour, la reconfiguration et la redistribution d'un manière auto-adaptative. Chaque machine ciblée doit contenir le DisnixService qui permet à la machine coordinatrice d'effectuer des opérations de redéploiement.

Les utilisateurs de Disnix peuvent être les administrateurs des systèmes orientés service distribués, mais plus généralement, plusieurs parties prenantes peuvent être impliquées dans l'expression du déploiement. Par exemple, des développeurs peuvent mettre en place

le *service model*. Les différents besoins sont bien séparés. Cependant, tous les utilisateurs de Disnix doivent être des experts dans leur propre domaine. En utilisant la version basique de Disnix, le plan de déploiement doit être explicitement fourni par le concepteur du déploiement.

3.3.2.4 Rapid Application Configurator

Problème Dans le contexte de l'informatique en nuage (ou *Cloud computing*), l'usage de *Virtual Appliances* (VA, appareils virtuels qui sont des images de machines virtuelles pré-empaquetées et des composants logiciels pré-installées) allège l'installation et la configuration de logiciels. Cependant, plusieurs problèmes résultant de la configuration embarquée dans les VAs sont encore présents : plusieurs images de machines virtuelles doivent être générées afin de couvrir les multiples combinaisons de composants logiciels correspondant aux multiples scénarios de déploiement. En plus, les interdépendances parmi les VAs, dans les applications multi-tiers, augmentent la complexité de la configuration du système et limitent les possibilités de personnalisation.

Solution Liu propose Rapid Application Configurator (RAC), une approche pour le développement logiciel, l'installation et la configuration basée sur la séparation des besoins et l'inversion de contrôle [1] [Liu, 2011]. Pour le déploiement, l'idée est de séparer les données de configuration de l'application logique, et de définir des propriétés configurables comme étant des variables dans l'entête du fichier de VA. Les variables doivent être fixées (et la machine virtuelle configurée) au moment du déploiement en utilisant les métadonnées de configuration. En pratique, lors du déploiement de VA configurable, le conteneur RAC (cf. figure 3.6) parse les métadonnées de configuration, effectue une validation initiale (vérification d'erreurs basiques), puis les instancie, les configure et lance la machine virtuelle. L'ensemble du processus est illustré dans la figure 3.7.

Afin d'être capable d'échanger des valeurs, le conteneur RAC et les machines virtuelles exposent les interfaces Web. Le conteneur RAC est implémenté comme étant un Web service lancé sur un serveur dédié et peut lire des valeurs à partir des machines virtuelles. Dans chaque machine virtuelle, un agent résidant réalise la configuration et la reconfiguration : il demande les valeurs de configuration une première fois au démarrage et puis périodiquement sonde ces valeurs pour vérifier si des changements ont été effectués.

Discussion RAC propose une solution spécifique au Cloud pour l'automatisation de la configuration et de la reconfiguration, ce qui permet de supprimer l'humain de la boucle. Cette solution est basée sur la séparation des besoins et l'inversion de contrôle.

La configuration d'une application est distribuée entre trois rôles : le producteur de VA, l'expert de la configuration et l'utilisateur final, chacune des parties ayant des connaissances requises et une certaine expertise. Cette proposition permet des réductions du coût et du

1. Haller *et al.* défini l'« inversion de contrôle » (IoC, *inversion of control*) comme suit [Haller and Odersky, 2006] : « *Au lieu d'appeler des opérations bloquantes (par ex. pour obtenir l'entrée d'un utilisateur), un programme enregistre simplement l'action à effectuer à certains évènements (par ex. un évènement signalant l'appui sur un bouton, ou bien le changement dans un champ textuel). Dans le processus, des gestionnaires d'évènements sont installés dans l'environnement d'exécution et sont appelés lorsqu'un évènement se produit. L'environnement d'exécution achemine les évènements aux gestionnaires installés. Ainsi, le contrôle à travers l'exécution d'un programme s'en retrouve « inversé ».* »

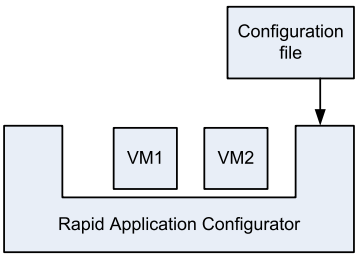


Figure 3.6 – Le conteneur RAC [Liu, 2011]

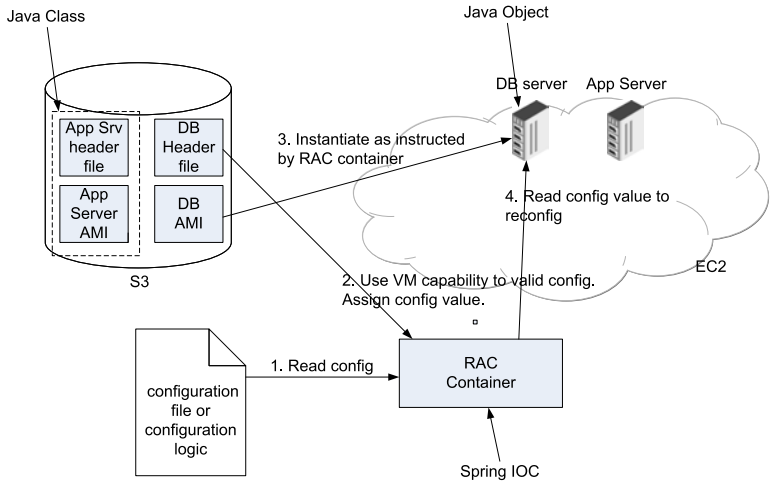


Figure 3.7 – Processus de déploiement basé sur RAC sur un Cloud Amazon EC2 [Liu, 2011]

temps pour la gestion de la configuration. Afin de relever l'humain des tâches de configuration, les métadonnées remplacent les installations manuelles et le conteneur RAC remplace l'utilisateur final. Le bootstrap nécessaire est la plateforme de virtualisation.

3.3.3 Allocation dynamique de ressources

Les grilles de calcul et les Clouds fournissent des environnements d'exécution pour du calcul haute performance à travers des ressources hétérogènes et des domaines administratifs multiples. Les outils de gestion associés fournissent des services pour gérer les ressources comme l'ordonnancement des tâches. Néanmoins, comme la disponibilité et la qualité des ressources n'est pas prévisible de manière permanente ou peut varier à l'exécution, l'adaptation dynamique du plan de déploiement est requise afin de fournir aux composants les ressources requises. Toutefois, le problème de placement qui consiste à trouver une localisation pour chacun des programmes est une tâche complexe. CoRDAGe s'intéresse au déploiement autonomique d'applications de longue durée d'exécution sur les grilles à grande échelle [Cudennec et al., 2008], et Wrangler cible le déploiement automatique d'applications distribuées dans le Cloud [Juve and Deelman, 2011a].

3.3.3.1 CoRDAGe

Problème Les applications pour grilles peuvent fonctionner des jours ou même des semaines sur des centaines ou des milliers de nœuds. Le besoin exact de ressources physiques est difficile à prévoir d'avance ou à l'initiation du déploiement. Ainsi, l'opérateur du déploiement doit surveiller les applications et satisfaire de manière permanente les exigences en ressources : en pratique, l'élasticité du domaine de déploiement est requise afin d'étendre ou rétracter l'application. Malheureusement, les outils de déploiement existants effectuent le déploiement en un coup, ils ne fournissent pas de support pour le (re)déploiement continu. Un autre problème concerne la gestion du « co-déploiement », c'est-à-dire le déploiement de plusieurs applications couplées.

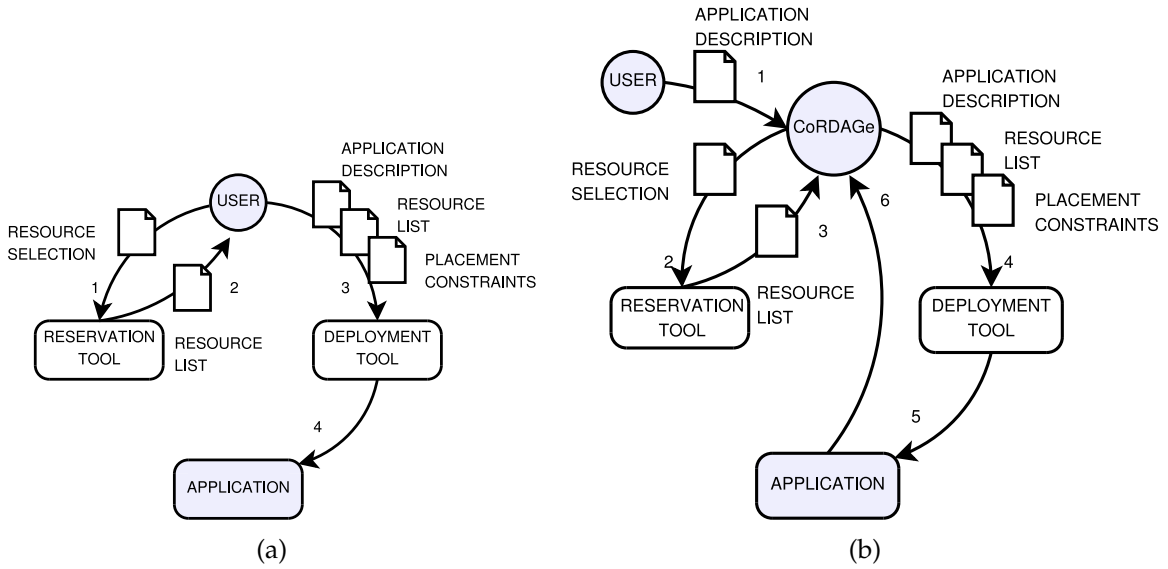


Figure 3.8 – Déployer une application à la main (a), et en utilisant l'outil CoRDAGe (b) [Cudennec et al., 2008]

Solution Cudennec *et al.* proposent CoRDAGe, un outil de co-déploiement et de redéploiement [Cudennec et al., 2008]. Il est basé sur ADAGE [ADA], un outil pour le déploiement centralisé et automatique sur les grilles de calculs. Les informations en entrée du système sont : une description de l'application, une description de la qualité de service à l'exécution, et une description des ressources ou de leur localisation. Lors d'une étape d'ordonnancement, un plan de déploiement est construit à partir de ces descriptions : les composants de l'application sont associés à un sous-ensemble de ressources sélectionnées, qui satisfont les contraintes concernant le système d'exploitation, la mémoire, etc. Puis, le plan de déploiement est réalisé : les fichiers (exécutables et fichiers de données) sont transférés en fonction du plan, des processus sont créés, configurés et lancés, et finalement un rapport de déploiement est généré.

CoRDAGe supporte le déploiement d'applications avant, pendant et après leur exécution. La figure 3.8b illustre les avantages de CoRDAGe : le déploiement est transparent, c'est-à-dire que l'utilisateur n'est pas en charge de la demande de ressources ni du déploiement des applications (comme illustré dans la figure 3.8a) ; la seule information requise de l'utilisateur est la description de l'application initiale.

Des modèles génériques de haut-niveau sont utilisés pour représenter en même temps l'application et les ressources physiques. Une application est décrite en tant qu'ensemble type d'entités *type of entities*, chacune étant un programme qui doit être exécuté sur une seule ressource physique (un nœud de calcul). La configuration consiste en la définition d'entités en instanciant leurs types, les entités étant les unités de déploiement de CoRDAGe. CoRDAGe génère des représentations sous forme d'arbre de l'application à partir de la description de l'application et des ressources physiques (le domaine de déploiement) en se basant sur ceux disponibles. Ensuite, une correspondance entre l'arbre de l'application et l'arbre physique décide du placement des entités ; ce placement (distribution du système) est effectué par l'outil de déploiement fourni par la grille en fonction d'un ensemble de contraintes à satisfaire.

À l'exécution, en se basant sur les services de CoRDAGe, les applications peuvent

gérer les opérations de déploiement autonomiquement en demandant une expansion ou une rétractation, sans aucune interaction avec l'utilisateur. L'expansion dynamique et la rétractation sont aussi supportées par les opérations sur les arbres.

Discussion Se basant sur ADAGE et des outils middleware de déploiement et de réservation, CoRDAGe gère le déploiement d'applications distribuées de longue durée dont la structure peut changer à l'exécution. Il se concentre sur le placement de programme (placement initial et redistribution) et leur configuration sur des grilles à grandes échelles dont la disponibilité des ressources et la qualité varient dynamiquement. CoRDAGe peut gérer plusieurs applications en même temps, en prenant en considération les contraintes spatiales et temporelles des applications interdépendantes.

Le bootstrap sur les appareils est la plateforme ADAGE (et le middleware de la grille) qui permettent un déploiement local. Le concepteur du déploiement doit spécifier l'application initiale en utilisant le modèle haut-niveau seulement, mais pas le domaine de déploiement. Par conséquent, une petite expertise dans le déploiement est demandée.

3.3.3.2 Wrangler

Problème Comparés aux grappes et aux grilles, les Clouds sont hautement dynamiques, spécialement lorsque plusieurs fournisseurs sont impliqués. Étant donné cette dynamique, déployer des services dans le Cloud en tant qu' *Infrastructure as a Service* (IaaS, infrastructure en tant que service) est un défi. Même si les infrastructures de Cloud permettent un approvisionnement de ressources, elles manquent de service de déploiement, de configuration et personnalisation spécifique aux applications des environnements d'exécution (c'est-à-dire pour construire les grappes virtuelles qui hébergent les applications). Ces tâches peuvent être faites manuellement, mais elles sont complexes, coûteuses en temps et sujettes aux erreurs.

Comme pour les applications de calcul haute performance, des outils sont nécessaires pour automatiser l'installation, la configuration et l'exécution des services distribués. Dans [Juve and Deelman, 2011a], les auteurs listent un ensemble d'exigences : déployer automatiquement des applications distribuées, supporter des dépendances complexes, permettre l'approvisionnement dynamique dans le but d'adapter le déploiement de l'application aux exigences à l'exécution, supporter plusieurs fournisseurs de Cloud, et gérer de manière continu l'état du déploiement.

Solution Juve *et al.* proposent un système appelé Wrangler qui permet aux utilisateurs de spécifier leurs applications de manière déclarative (*via* une description XML), et d'automatiquement approvisionner, configurer et contrôler leur déploiement sur les Clouds IaaS [Juve and Deelman, 2011a]. Wrangler s'interface avec plusieurs fournisseurs de Cloud afin d'approvisionner des machines virtuelles, coordonner la configuration et l'initiation de services pour supporter les applications distribuées, et contrôler les applications à travers le temps. Il supporte les possibles pannes, et l'ajout et la suppression dynamique de nœuds.

L'architecture distribuée de Wrangler est représentée à la figure 3.9. Il est basé sur quatre types de composants : *client*, *coordinateur* (*coordinator*), *agent*, et *module* (*plugin*). Les clients s'exécutent sur les machines des utilisateurs et envoient des requêtes au coordinateur (unique) pour un nouveau déploiement, un déploiement incrémental ou une terminai-

son. Les requêtes pour le déploiement incluent des descriptions XML des nœuds à lancer, des images de machines virtuelles et des modules à utiliser. Les modules sont des scripts utilisateur qui définissent modulairement différents aspects du comportement spécifique à l'application pour un nœud (la configuration). Le coordinateur est un WebService : il est un intermédiaire entre les clients et les agents. Il interagit aussi avec le fournisseur de ressources afin d'approvisionner les machines virtuelles adéquates. Tous les nœuds du domaine de déploiement hébergent un agent : le code de l'agent est pré-installé dans l'image de la machine virtuelle, et quand la machine virtuelle est lancée, elle lance l'agent qui s'enregistre auprès du coordinateur. Les agents reçoivent des commandes de configuration : en réponse, ils récupèrent la liste des modules du nœud à partir du coordinateur, les téléchargent et les lancent. En même temps, les agents contrôlent le nœud qui les héberge en invoquant la méthode « status » (statut) des modules, et envoie les données collectées au coordinateur. Ils mettent fin aussi au module en réponse à une commande de terminaison. Par conséquent, les agents sont en charge des tâches de déploiement distribué et décentralisé.

Figure 3.9 – Architecture du système Wrangler [Juve and Deelman, 2011a]

Discussion Wrangler est une solution partiellement décentralisée pour le déploiement sur des Clouds dynamiques, ayant plusieurs fournisseurs d'accès, d'applications distribuées dont les exigences de ressources peuvent varier à l'exécution. Les composants de l'application sont des images de machines virtuelles. Afin d'effectuer ce déploiement, le concepteur du déploiement doit spécifier les composants avec leurs paramètres et leurs dépendances, les fournisseurs de Cloud et les modules. Le concepteur et l'opérateur de déploiement doivent avoir une certaine expertise, d'autant qu'ils peuvent être impliqués dans la construction d'images de machine virtuelle ou la correction de problèmes à l'exécution.

3.3.4 Gestion des appareils mobiles et à ressources limitées

La mobilité des appareils à capacité limitée pose des problèmes liés à la déconnexion et la qualité de service, ce qui demande un déploiement à la volée. Kalimucho se concentre sur l'adaptation du plan de déploiement à la qualité de service [Louberry et al., 2011b],

StarCCM supporte le déploiement sensible au contexte [Zheng et al., 2006] et Codewan supporte le déploiement opportuniste de logiciels à base de composants sur les Mobile Ad hoc NETworks (MANET) déconnectés [Guidec et al., 2010]. Cloudlet est une solution à base de machines virtuelles qui permet de transférer la charge des appareils mobiles sur un Cloud de proximité [Satyanarayanan et al., 2009].

3.3.4.1 Kalimucho

Problème Les appareils mobiles à ressources limitées amènent un nouveau défi auquel les plateformes de déploiement doivent faire face. Afin de garantir une qualité de service suffisante aux utilisateurs finaux, la distribution (l'exécution du plan de déploiement) doit s'adapter à l'environnement de manière réactive et dynamique.

Solution Kalimucho est une plateforme distribuée pour le déploiement dynamique et la reconfiguration sur des appareils hétérogènes comme des ordinateurs de bureau, des ordinateurs portable et des appareils mobiles [Cassagnes et al., 2009]. Les applications sont composées de composants métier liés par des connecteurs, respectant le modèle de composant Osagaia et le modèle de connecteur Korrontea. En accord avec le principe de la séparation des préoccupations, les composants métier d'Osagaia fonctionnent à l'intérieur d'un conteneur configurable qui supporte la connexion aux autres conteneurs en utilisant les connecteurs Korrontea. L'un des services de Kalimucho est dédié à la supervision et la distribution à travers le domaine de déploiement. Il a une vision globale du réseau et des appareils. D'autres services basiques permettent la création, l'arrêt et le retrait de composants ou de conteneurs, la migration de composants, leurs connexion et déconnexion, etc.

Louberry *et al.* introduisent une heuristique de déploiement contextuel, afin de sélectionner une configuration qui satisfasse les exigences de qualité de service, et ainsi mieux placer les composants sur les appareils [Louberry et al., 2011b]. Les paramètres de l'algorithme sont les types des appareils, des mesures d'énergie, de charge du processeur et de mémoire disponible. Par conséquent, quand il y a une nouvelle exigence fonctionnelle (par ex. une action de l'utilisateur final), si un appareil apparaît ou disparaît du réseau ou dont les ressources deviennent basses, ou si la priorité est le changement d'exigences, la plateforme Kalimucho met à jour le plan de déploiement et le réalise. La reconfiguration et la redistribution sont effectuées pendant que l'application est en exécution sans avoir à l'arrêter, ainsi, la continuité du service et la durabilité des applications sont assurées.

Discussion Kalimucho permet une adaptation décentralisée et sensible au contexte du plan de déploiement. Les applications déployées sont un système de composants Java qui respectent les modèles de composant Osagaia et Korrontea. Kalimucho supporte le déploiement d'activités sur le site consommateur : l'installation et la désinstallation, la mise à jour, la reconfiguration et la redistribution. Les appareils du domaine de déploiement doivent héberger la plateforme Kalimucho (probablement une version limitée en fonction des capacités des appareils) ; ils peuvent dynamiquement entrer et quitter le domaine. Les utilisateurs peuvent interagir avec la plateforme Kalimucho en mettant en place un plan de déploiement ou plus explicitement en demandant l'activation ou le déplacement d'un composant. Ils peuvent aussi exprimer des propriétés requises de qualité de service, et des paramètres de l'heuristique de déploiement en décrivant les composants (contraintes et exigences), les configurations, les appareils, etc. Par conséquent, le concepteur du déploiement

doit être expert dans la technologie Kalimucho.

3.3.4.2 StarCCM-based Deployment

Problème D'un côté, les applications ubiquitaires doivent être sensibles aux ressources disponibles et au changement constant de contexte à l'exécution, et donc être capable de s'adapter. L'automatisation du déploiement est ainsi requise. D'un autre côté, le middleware à base de composants comme Corba Component Model (CCM) fournit des facilités de déploiement mais non sensibles au contexte. Ici, le problème concerne l'adaptation à la volée du plan de déploiement au contexte d'exécution.

Solution Zheng *et al.* proposent une approche basée sur du middleware pour le déploiement d'applications distribuées à base de composants et sensibles au contexte [Zheng et al., 2006]. L'implémentation des applications et des middleware sont basés sur CCM.

L'architecture générale du middleware (cf. figure 3.10) est basée sur trois services centraux : la gestion de contexte qui fournit des informations à propos du contexte, la gestion de l'adaptation qui décide d'un plan de déploiement, et la gestion de la configuration qui réalise le plan, c'est-à-dire la reconfiguration et la redistribution.

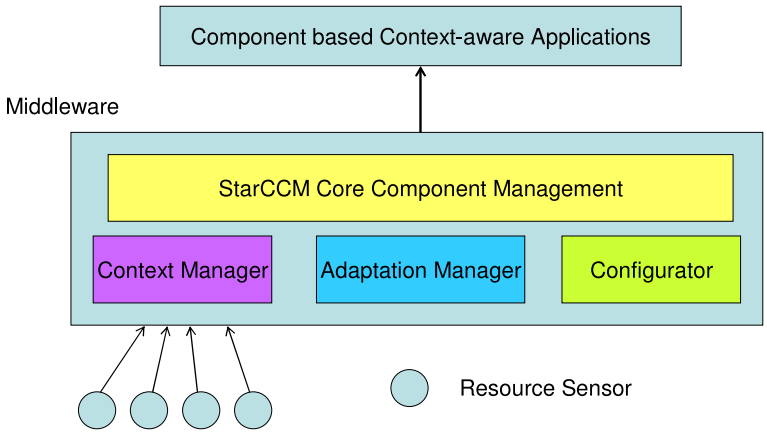


Figure 3.10 – Architecture du middleware sensible au contexte [Zheng et al., 2007]

La figure 3.11 explique comment les informations de contexte sont collectées, filtrées et traitées, et présente le *component deployer* (déployeur de composant, qui est une abstraction pour les composants adaptateur et configurateur). La chaîne de traitement des données est basée sur un patron de *publish/subscribe*, et sur différents composants : agents capteurs, collecteurs, interpréteurs et analyseurs.

Le noyau de la solution est l'*Adaptation Manager* (gestionnaire d'adaptation). Il calcule un nouveau plan de déploiement à l'exécution, en se basant sur un ensemble de règles à propos de l'unité d'un composant (sélection de version, adaptation individuelle par configuration) ou d'une application (adaptation au niveau de l'assemblage, composants optionnels, placement) : un algorithme $A*$ sélectionne les meilleures versions et appareils pour toutes les instances de composants.

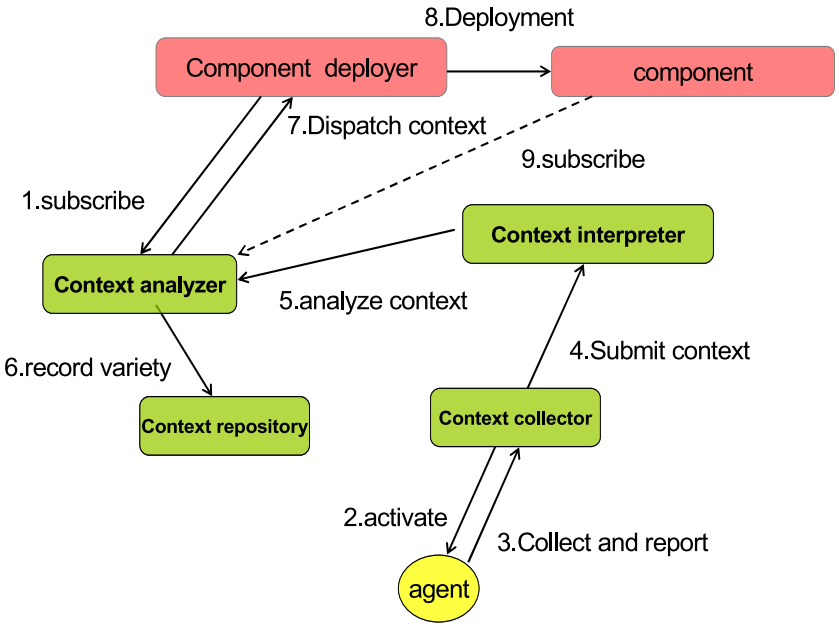


Figure 3.11 – Dynamique du middleware sensible au contexte [Zheng et al., 2006]

Discussion Zheng *et al.* se concentrent sur la gestion de contexte comme élément clé pour le déploiement adaptatif. StarCCM supporte le déploiement à la volée et sensible au contexte d'applications distribuées à base de composants à travers des réseaux d'appareils dont les ressources sont limitées et changeantes. L'architecture est abstraite mais son implémentation est rattachée à CCM. La solution vise l'adaptation dynamique du plan de déploiement, et dont le middleware supporte les activités de reconfiguration et de redistribution. Dans ce contexte, le concepteur du déploiement doit spécifier les règles d'adaptation au niveau du composant et du système. Il doit être un expert dans le déploiement et aussi avoir une connaissance à propos des composants qu'il déploie.

3.3.4.3 Codewan

Problème Les réseaux de capteurs autonomes (*MANETs*), avec leur instabilité, déconnexions et fragmentation sont le cœur du projet SARAH [SAR]. Dans les réseaux d'infrastructure, le déploiement de composants logiciels (et plus précisément leur distribution) est basé sur un serveur hôte qui stocke les composants dans des dépôts de composants et les délivre à la demande. Cependant, dans les réseaux MANETs, les composants ne peuvent être récupérés d'un dépôt central. De plus, à cause des changements dans la structure du réseau continus et imprévisibles, il n'y a aucune garantie que la demande puisse être satisfaite car le composant peut être (temporairement ou non) inaccessible dans le voisinage demandeur.

Solution Guidec *et al.* présentent un modèle coopératif décentralisé pour la mise à disposition et la distribution de composants logiciels sur les réseaux MANETs déconnectés, et une implémentation d'un middleware nommé Codewan [Guidec et al., 2010]. Dans leur voisinage, les appareils interagissent de manière opportuniste afin de découvrir et d'échanger les composants logiciels.

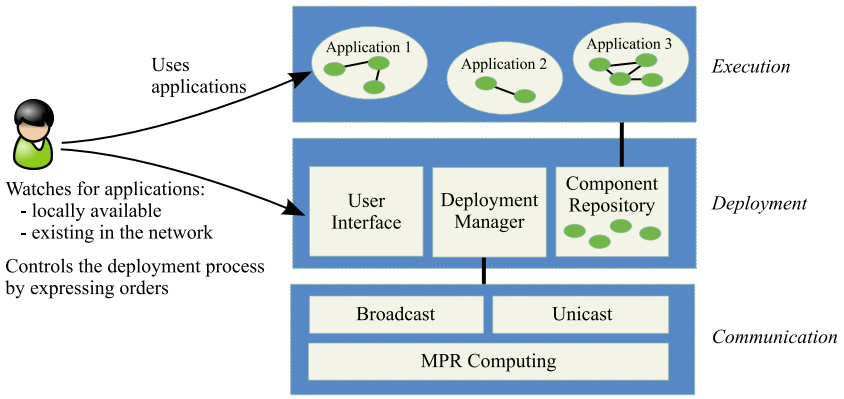


Figure 3.12 – Architecture de la plateforme Codewan [Guidec et al., 2010]

Codewan est composée de trois couches représentées dans la figure 3.12. La couche de communication opportuniste supporte la dissémination de composants : leurs annonces sont multi-diffusées tandis que des transmissions mono-diffusion supportent les demandes en réponse. Sur un appareil, la couche de déploiement contient un composant de dépôt local, un gestionnaire de déploiement et une interface graphique. L'interface utilisateur permet à l'utilisateur d'observer le statut du composant et de demander (ou annuler) le déploiement d'un composant, ou d'un application à base de composants. Le gestionnaire de déploiement prend ses ordres directement de l'utilisateur final. La coopération pair à pair entre les gestionnaires de déploiement permet aux appareils d'obtenir des copies des composants logiciels requis par l'utilisateur et disponibles dans le voisinage, tandis que, symétriquement, le voisinage peut bénéficier des composants disponibles à partir du même appareil. Le gestionnaire de déploiement est responsable de la mise à jour du dépôt local. Il peut gérer complètement le déploiement (local) d'une application à base de composants : il peut examiner les dépendances entre les composants et lancer un processus de récupération complet. Afin d'effectuer ces opérations efficacement, il apprend de part les interactions qu'il a avec son voisinage.

Discussion Codewan vise le déploiement d'applications à base de composants (sans être lié à un modèle de composant en particulier) sur les MANETs déconnectés. Le déploiement est local et à la demande, et supporte parfaitement la dynamique des MANETs. Codewan se concentre sur une partie de l'activité d'installation, la distribution. De plus, cette solution se base sur un modèle pour l'empaquetage de composants, et entre dans le domaine de l'activité de dépôt. La plateforme Codewan est implémentée en Java et doit être pré-installée sur tous les appareils impliqués. Le déploiement est dirigé par l'utilisateur du logiciel, qui est aussi l'utilisateur de l'appareil, en utilisant une interface graphique qui requiert quelques compétences avancées. Cependant, pour l'empaquetage, le producteur du logiciel doit être un expert.

3.3.4.4 Cloudlet

Problème La plupart des appareils mobiles sont intrinsèquement limités en ressources. Cela peut être une limitation majeure pour des applications avancées qui requièrent des ressources comme de la puissance de traitement ou de l'énergie. Une solution peut être trouvée dans les Cloud, mais les interactions dans un réseau étendu (type WAN, *Wide Area Network*)

soulèvent des problèmes de latence, de longs délais et coupures. Satyanarayanan *et al.* introduisent le concept de Cloudlet, pour décrire un Cloud local [Satyanarayanan et al., 2009]. Au lieu de déléguer la tâche de calcul à un Cloud distant, les auteurs proposent de le déléguer à un appareil proche (accessible dans un réseau local) et riche en ressources (le Cloudlet), évitant ainsi certains problèmes.

Solution Satyanarayanan *et al.* présentent une solution basée sur la « personnalisation éphémère » (*transient customization*) de l'infrastructure de Cloudlet en utilisant la technologie des machines virtuelles et la synthèse dynamique de machines virtuelles (cf. figure 3.13) : une machine virtuelle personnalisée est dynamiquement synthétisée à partir d'une machine virtuelle de base, *base VM*, pré-chargée sur l'infrastructure de la Cloudlet et une petite surcouche de machine virtuelle complémentaire qui encapsule l'application, et qui est transférée de l'appareil mobile à la Cloudlet. La machine virtuelle de base est étendue au moyen d'un script pour l'installation et la reprise. Après avoir lancé l'application à distance dans une machine virtuelle personnalisée, cette dernière est détruite et la Cloudlet retourne à son état initial. Par conséquent, les performances ne dépendent que des ressources locales (bande passante entre la Cloudlet et l'appareil, la puissance de calcul de la Cloudlet, etc.) et les pannes du réseau étendu n'affectent ni la synthèse ni l'exécution.

Une implémentation, nommée Kimberley, est basée sur une gestionnaire de machines virtuelles pour Linux. Le problème principal est la taille et la sécurité des Cloudlet.

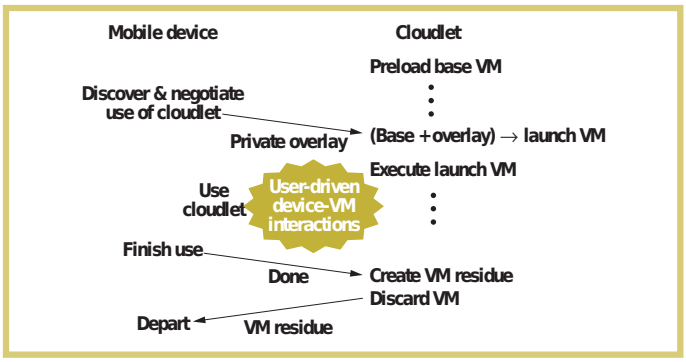


Figure 3.13 – Chronologie de la synthèse de machines virtuelles dynamiques [Satyanarayanan et al., 2009]

Discussion La solution Cloudlet déploie des applications entières indépendamment de la technologie sur un seul site distant (la Cloudlet) qui apparaît spontanément dans un contexte de mobilité. Les activités gérées sont l'installation et la désinstallation (en incluant le transfert), l'activation et la désactivation. Le transfert est initié par l'utilisateur final dont l'expertise peut-être faible. Afin de fonctionner, une *Kimberley Control Manager* (KCM) doit être activée sur la Cloudlet et l'appareil mobile, et la Cloudlet doit aussi héberger une machine virtuelle de base.

3.3.5 Formalisation de haut-niveau et expressivité

Concevoir un déploiement prenant en compte un grand nombre de composants et d'appareils est un défi. Les concepteurs doivent prendre en considération des activités

différentes mais reliées et les contraintes du domaine de déploiement, de l'application et ses composants, et les exigences qui peuvent provenir des différentes parties prenantes. Cependant, exprimer directement un plan de déploiement n'est pas toujours souhaitable ou même possible, par exemple quand les appareils du domaine ne peuvent être connus qu'au moment de l'exécution.

Cette section étudie différents travaux qui se focalisent sur l'abstraction et la facilitation de l'expression du déploiement. ORYA propose un framework pour l'expression de stratégies de déploiement à base de propriétés [Cunin et al., 2005]. DeployWare est un framework complet pour le déploiement à grande échelle basé sur un langage de modélisation dédié au déploiement [Flissi et al., 2008a]. ADME est un autre framework qui vise le déploiement autonomique et se base sur une résolution de problème de satisfaction de contraintes [Dearle et al., 2004]. TUNe fournit des formalismes de haut-niveau pour la gestion autonomique de grilles à grande échelle décentralisées [Broto et al., 2008, Toure et al., 2010]. SmartFrog a été conçu avec pour objectif d'effectuer la conception, le déploiement et la gestion de systèmes distribués à base de composants de manière plus simple et plus robuste [Sabharwal, 2006, Goldsack et al., 2009].

De plus, comme la conception du déploiement est une opération particulière avec des exigences spécifiques, certains langages dédiés ont été proposés pour satisfaire le besoin d'expression des propriétés de déploiement. Les DSLs permettent une définition des propriétés de déploiement en utilisant des idiomes et abstraction, ainsi ils peuvent être utilisés efficacement par les experts du domaine [Strembeck and Zdun, 2009]. Parmi les DSLs pour le déploiement, nous pouvons citer Deladas (cf. section 3.3.5.3) [Dearle et al., 2004], J-ASD [Matougui and Leriche, 2012], et Pim4Cloud [Brandtzæg et al., 2012]. Dans [Sledziewski et al., 2010], les auteurs recommandent l'usage de DSL afin d'améliorer le développement d'application et le déploiement sur le Cloud. Néanmoins, étudier les DSLs pour le déploiement est en dehors du cadre de cet état de l'art, et nous en proposerons une courte étude plus loin dans le chapitre 5.

3.3.5.1 ORYA

Problème Le déploiement d'applications sur un grand ensemble de machines est complexe et ne peut être réalisé à la main. Des stratégies avancées implémentées par des bricolages maisons (scripts) sont souvent utilisées dans les entreprises par les administrateurs système. Les difficultés proviennent de l'expression et de l'implémentation.

Solution Cunin *et al.* proposent un framework appelé ORYA (Open enviRonment to deploY Applications) pour la conception et pilotage de stratégies de déploiement spécialisées basées sur des propriétés [Cunin et al., 2005].

Une stratégie est attachée à une machine ou à un groupe de machines et est appliquée à un ensemble d'unités déployables. Elle peut concerner la sélection d'unités de déploiement et leurs versions, l'ordre des opérations de déploiement, ou la prise en compte des exceptions, et prend en compte les propriétés et contraintes du domaine de déploiement et de l'application. Elle est définie d'une manière déclarative par un triplet : <*Activité, ExpressionLogique, Choix*> (<*Activity, LogicalExpression, Choice*>). L'*Activité* spécifie une activité de déploiement telle que définie dans la section 2.2.3 (mais est limitée à l'installation et la mise à jour). Lorsque l'activité est effectuée, l'*ExpressionLogique* est évaluée pour toutes les unités déployables, les divisant en deux sous-ensembles : l'« ensemble vrai » et l'« ensemble

faux » (respectivement, "true set" et "false set"). Le *Choix* définit les règles pour l'exécution de l'activité sur chaque sous-ensemble. En plus du comportement basique, une stratégie est définis par certaines caractéristiques comme le cadre (lié au domaine), la visibilité et la précédence afin d'éviter des ambiguïtés en cas de stratégies concurrentes, etc.

Discussion ORYA supporte l'expression de stratégies de déploiement, en particulier celles concernant les choix et l'ordre des opérations sur le domaine de déploiement. Essentiellement, le domaine est un réseau local d'une entreprise, qui peut être divisé en sous-domaines. Les activités concernées sont principalement l'installation et la mise à jour. Le plan de déploiement n'est pas explicité, mais le domaine et les unités de déploiement doivent l'être, avec leurs dépendances, contraintes et règles de déploiement. Ainsi, le concepteur doit être un expert en déploiement.

3.3.5.2 DeployWare

Problème Flissi *et al.* analysent les principaux défis du déploiement à grande échelle sur des grilles. Les frameworks de déploiement doivent prendre en compte les problèmes dus à la complexité résultant du grand nombre de nœuds et des dépendances logiciels, l'hétérogénéité du logiciel et du domaine de déploiement, la fiabilité, la parallélisation, le passage à l'échelle, et le contrôle [Flissi et al., 2008a].

Solution Flissi *et al.* proposent DeployWare, un framework générique pour le déploiement de systèmes logiciels distribués et hétérogènes sur des grilles [Flissi et al., 2008a]. DeployWare fournit un langage dédié de modélisation (DSML, *Domain-Specific Modeling Language*) basé sur un métamodèle qui capture les concepts abstraits du déploiement, indépendamment du paradigme du logiciel et de la technologie. Le modèle de DeployWare décrit les configurations à déployer. Elles sont écrites en utilisant un langage de description d'architecture (ADL, *Architecture Description Language*), et validées avant l'exécution afin d'en assurer la fiabilité.

L'exécution de DeployWare est constituée de composants logiciels Fractal et distribués sur un ensemble de nœuds serveurs sélectionnés. Ils exécutent les descriptions DeployWare : une machine virtuelle interprète les descriptions et orchestre automatiquement un processus de déploiement complexe, en prenant en compte les dépendances logicielles et l'hétérogénéité matérielle (cf. figure 3.14). De plus, une console graphique permet aux administrateurs de contrôler et gérer le système déployé à l'exécution.

Discussion DeployWare est une solution complète avec des outils intégrés, qui vise le déploiement de systèmes logiciels distribués sur des réseaux à grandes échelles. Cependant, elle ne répond pas aux problèmes des environnements instables ou ouverts. DeployWare amène le déploiement vers un haut-niveau d'abstraction, en utilisant un métamodèle et un ADL, dissimulant ainsi au développeur la complexité de l'orchestration du déploiement. Plusieurs parties prenantes peuvent contribuer séparément à l'expression du déploiement : les administrateurs système, les experts du logiciel, les utilisateurs du logiciel (les utilisateurs définissent le plan de déploiement). Dû à la séparation des besoins, chacun d'eux exprime des propriétés de déploiement en se basant sur sa compétence métier. À l'exécution, DeployWare doit être présent sur les machines cibles afin de gérer localement le déploiement.

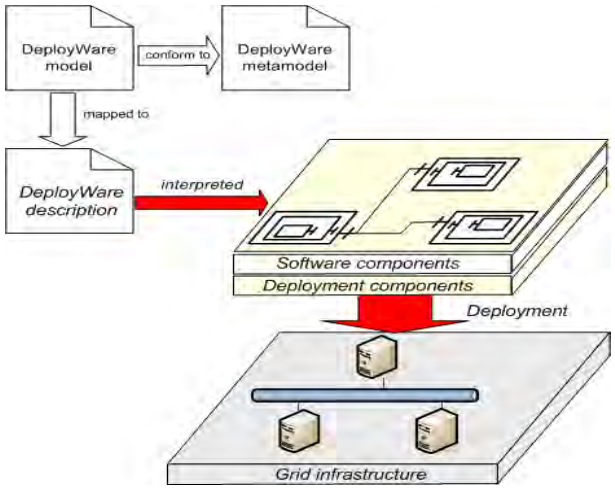


Figure 3.14 – Architecture de DeployWare [Flissi et al., 2008a]

3.3.5.3 ADME

Problème Le déploiement d'applications distribuées à base de composants pose deux problèmes majeurs : la conception du déploiement initial et son évolution à l'exécution lorsqu'il doit faire face à des pannes d'hôtes ou d'autres incidents. Comme ces problèmes sont trop complexes pour être gérés par un opérateur humain, Dearle *et al.* ont pour but d'appliquer une boucle autonomique définies par Kephart *et al.* [Kephart and Chess, 2003].

Solution Dearle *et al.* proposent ADME (*Autonomic Deployment and Management Engine*, moteur de déploiement et de gestion autonomique du déploiement), un framework pour le déploiement et la gestion autonomique d'applications distribuées à base de composants [Dearle et al., 2004]. ADME utilise Deladas (*Declaratory Language for Describing Autonomic Systems*, langage déclaratif pour la description de systèmes autonomiques), un DSL qui supporte l'expression de propriétés de déploiement en utilisant des contraintes. Les contraintes sont traitées par un solveur de contraintes qui génère une configuration (un plan de déploiement) encodé en XML. Un moteur de gestion autonomique gère à la fois le déploiement initial, et adapte l'application aux changements de conditions à l'exécution. Pour approvisionner la boucle autonomique, les applications sont instrumentalisés avec des sondes qui gèrent localement l'exécution et génèrent des évènements. Les évènements sont collectés et traités par un composant central. En cas de panne d'un hôte par ex., ce composant essaie de trouver une nouvelle configuration et orchestre l'ensemble du processus d'adaptation. Excepté pour des solutions triviales, il doit lancer une fois de plus le solveur de contraintes afin de trouver un autre plan de déploiement. Cependant, afin de minimiser le redéploiement, le problème est plus contraint que l'originel : la partie du plan de déploiement courant qui n'est pas impliquée dans la situation à corriger doit être donnée en contrainte. Après cela, ces contraintes sont progressivement supprimées jusqu'à ce qu'une configuration soit trouvée.

Discussion ADME se concentre sur l'autonomie des systèmes en charge du déploiement dans un contexte de volatilité des ressources. Le langage Deladas fournit un haut-niveau d'expressivité pour les administrateurs d'applications distribuées à base de composants. Ainsi, Dearle *et al.* proposent une solution pour la gestion autonomique et la reconfiguration

et redistribution à l'exécution, qui ne requiert aucune intervention humaine. Le contrôle est décentralisé, mais à l'exécution, le calcul de nouvelles configurations est centralisé et un solveur de contraintes centralisé est requis pour les pires cas. Le système d'exécution d'ADME se base sur un middleware spécifique, appelé Cingal, qui doit être installé sur chaque machine du domaine de déploiement.

3.3.5.4 TUNe

Problème Les environnements de grilles de calculs sont à grande échelle, dynamiques et hétérogènes, et par conséquent complexes. Le déploiement et la gestion centralisée des applications distribuées n'est pas faisable : la décentralisation et l'autonomie pour la gestion de l'exécution sont requises. De plus, afin de décrire les propriétés de déploiement, il y a un besoin d'expressivité.

Solution Toure *et al.* proposent TUNe, un système de gestion autonomique dédié au déploiement décentralisé à grande échelle sur les environnements de grille [Broto et al., 2008, Toure et al., 2010]. Les composants logiciels sont enveloppés dans des composants Fractal, en utilisant un langage de description d'enveloppement (nommé WDL) qui permet de spécifier l'interface du composant fourni à la gestion de l'exécution de TUNe. Une instance TUNe est déployée sur chaque machine et le système de gestion est organisé hiérarchiquement pour déployer efficacement les composants Fractal. Sur les machines, des sondes spécifiques génèrent des évènements qui déclenchent des reconfigurations. Un évènement est traité en local, à moins qu'il ne concerne une entité gérée par une autre machine. Dans ce cas-là, il est transmis suivant un diagramme de déploiement.

De plus, des profils UML supportent la description de l'application, une représentation arborescente du domaine (c'est-à-dire l'environnement de la grille), la description de la politique d'administration décentralisée, et un diagramme d'état qui définit le *workflow* (flux de travaux) de démarrage et de reconfiguration de l'application. Le déploiement est basé sur une correspondance entre le diagramme de domaine, le diagramme d'administration et le diagramme d'application.

Discussion TUNe gère les applications distribuées à base de composants sur les grilles hétérogènes et dynamiques à grande échelle, en utilisant le modèle de composants Fractal et le middleware de grilles DIET. Les activités de déploiement prises en compte sont l'installation (incluant la distribution et la configuration), l'activation et la reconfiguration. Le plan de déploiement résulte d'une correspondance entre des modèles, et ces modèles doivent être fournis par des experts. À l'exécution, des sondes spécifiques gèrent le domaine de déploiement et les reconfigurations sont effectuées autonomiquement par des instances TUNe, sans requérir une participation humaine. Afin d'être capable de fonctionner, DIET et TUNe doivent être disponibles sur chaque machine.

3.3.5.5 SmartFrog

Problème La gestion de systèmes de composants distribués à base de composants sur des infrastructures de grilles (composées de nombreuses machines distribuées hétérogènes) manque de simplicité et de robustesse. Déployer manuellement des applications sur la grille

ne tient pas l'échelle avec le développement des grilles. De plus, comme différents frameworks peuvent être impliqués pour la configuration, la gestion du cycle de vie du composant, ou la gestion des erreurs, les données de déploiement provenant des différentes parties prenantes (développeurs des composants, intégrateurs, gestionnaires de déploiement, etc.) peuvent être dispersées, menant à des inconsistances et des incompréhensions.

Solution Sabharwal *et al.* et Goldsack *et al.* proposent le framework SmartFrog pour le déploiement dirigé par la configuration des infrastructures de grilles [Goldsack et al., 2009]. SmartFrog permet à l'utilisateur de décrire et configurer un système logiciel distribué comme étant un ensemble de composants coopérants, en utilisant un modèle spécifique [Sabharwal, 2006]. SmartFrog supporte l'installation automatique, l'activation et la gestion de composants qui sont déployés sur les machines virtuelles Java (*Java Virtual Machine*, JVM). L'infrastructure de déploiement de SmartFrog est basée sur des composants et distribuée, en utilisant Java RMI pour les interactions distantes. Ses composants fournissent des services pour la distribution et la maintenance des systèmes logiciels à l'exécution (gestion du cycle de vie du composant et des pannes). Essentiellement, un démon SmartFrog doit être lancé sur chaque machine du domaine de déploiement. Initialement les démons sont déployés d'une manière centralisée à partir d'une machine maître en utilisant des protocoles comme ssh ou ftp. Ensuite, l'infrastructure de déploiement est déployée en utilisant les démons, et finalement le système logiciel.

SmartFrog offre un framework pour l'expression de données de configuration d'une manière consistante en utilisant un modèle hiérarchique et des maquettes. Il fournit aussi des facilités pour l'établissement tardif de liens et une adaptation au contexte. SmartFrog permet aussi la définition de gestionnaire de cycle de vie pour la configuration. Les gestionnaires de cycle de vie configurent correctement les composants ou groupes de composants en fonction des données de configurations. Chacun d'eux implémente un ensemble de méthodes afin de pouvoir prendre en compte, par exemple, l'installation, l'activation, la désinstallation, la notification de pannes ou la vérification de statut.

SmartFrog fournit quelques autres fonctionnalités avancées. Les concepteurs du déploiement peuvent exprimer le plan de déploiement en donnant des valeurs de localisation exactes mais aussi des propriétés de localisation comme à proximité d'un composant ou d'un sous-système. Le déploiement peut être dynamiquement reconfiguré et adapté aux changements de conditions. Concernant la fiabilité, les structures de validation peuvent être incluses dans les configurations afin de vérifier des hypothèses particulières. De plus, afin de prévenir les attaques, SmartFrog propose un modèle de sécurité basé sur les domaines de sécurité et les autorités de certification.

Discussion SmartFrog est un framework de déploiement avancé, dédié aux infrastructures de grille. Il gère les déploiements de systèmes de composants distribués, et est basé sur un modèle de composants spécifique. Comme les systèmes évoluent, il est possible d'ajouter ou de supprimer des composants à l'exécution. SmartFrog gère différentes activités du cycle de vie : installation et désinstallation, activation et désactivation, adaptation et reconfiguration. Aucun bootstrap n'a besoin d'être initialement présent sur les machines. Les utilisateurs de SmartFrog utilisent un langage pour la description d'ensembles de composants et de paramètres de configuration de composants, avec des mécanismes pour la composition et l'extension. Ils doivent être experts en configuration et en gestion de cycle de vie.

3.4 Synthèse

Cet état de l'art montre que l'automatisation du déploiement logiciel a été au cœur de plusieurs travaux de recherches récents.

Cette section fournit une synthèse de cette étude, elle est organisée en quatre parties correspondant aux questions principales concernant le déploiement (cf. section 3.2). Pour chaque question, notre analyse est illustrée dans un tableau qui met en évidence la couverture des solutions étudiées.

3.4.1 Logiciel déployé

Les applications visées sont divisées en deux groupes : les applications monolithiques et les applications à base de composants. La plupart des solutions visent les applications à base de composants, comme la table 3.1a le montre. La moitié d'entre elles sont dédiées à un type particulier de composant logiciel, tandis que l'autre moitié ne fait aucune hypothèse vis-à-vis ce point (cf. table 3.1b). Cependant, parmi les dernières solutions, certaines requièrent que le composant soit enveloppé dans un type de composant spécifique, comme TUNe le fait en enveloppant ses composants dans des composants Fractal avant de les déployer. De plus, Disnix et DeployWare prennent en compte l'hétérogénéité du logiciel en utilisant des modèles et des transformations de modèle.

Nous observons aussi que la dynamique du système logiciel n'est pas prise en compte à part pour CoRDAGe, Wrangler et SmartFrog, ou bien limitée à la mise à jour (Software Dock, Disnix) ou la reconfiguration (RAC). En dernier lieu, seuls CoRDAGe, DeployWare, et SmartFrog prennent en compte la scalabilité du système logiciel.

	Application monolithique	Application à base de composants
FROGi	✓	
R-OSGi		✓
Soft. Dock	✓	
QUIET	✓	
Disnix		✓
RAC		✓
CoRDAGe		✓
Wrangler		✓
Kalimucho		✓
StarCCM		✓
Codewan		✓
Cloudlet	✓	
ORYA	✓	
DeployWare		✓
ADME		✓
TUNe		✓
SmartFrog		✓

(a) Nature de l'unité de déploiement

	Technologie
FROGi	Fractal
R-OSGi	OSGi
Soft. Dock	*tous*
QUIET	SUIPM, applications MS Windows
Disnix	*tous*
RAC	images de VM
CoRDAGe	*tous*
Wrangler	images de VM, scripts de configuration
Kalimucho	Osagaia - Korrontea
StarCCM	CCM
Codewan	*tous*
Cloudlet	*tous*
ORYA	*tous*
DeployWare	*tous*
ADME	*tous*
TUNe	*tous*
SmartFrog	modèle de composants SmartFrog

(b) Technologie de l'unité de déploiement

Tableau 3.1 – Unité de déploiement

3.4.2 Domaine de déploiement

La table 3.2a montre les différentes natures du domaine visées par les solutions de déploiement : localement sur une machine (avec contrôle distant), un réseau réparti, un

réseau spontané, un grille, ou le Cloud. La plupart des travaux s'intéressent à un domaine de déploiement constitué de machines reliées par un réseau, qui peut être particulier comme dans le cas des infrastructures de grilles ou de clouds, voir la table 3.2a. Les problématiques principalement visées sont l'hétérogénéité (Disnix, DeployWare, TUNe, SmartFrog), la limitation des ressources de la machine (Kalimucho, Cloudlet), et la dynamique du domaine, c'est-à-dire les connexions et les déconnexions (spécialement pour les réseaux spontanés), les pannes de machines et des liens de communication, les variations de qualité de services, etc. La table 3.2b indique si et comment ces solutions prennent en compte la dynamique du domaine de déploiement. À part pour TUNe, aucune de ces propositions ne gère l'hétérogénéité, la scalabilité et la dynamique en même temps.

Les solutions ne prennent pas en compte l'administration multiple au sein des domaines et le plus souvent supposent que le système de déploiement a la permission d'effectuer des opérations sur les machines. L'exception est SmartFrog, qui propose un modèle de sécurité particulier.

	Local (distant)	Réseau	Réseau spontané	Grille	Cloud
FROGi	✓				
R-OSGi		✓			
Soft. Dock		✓			
QUIET		✓			
Disnix		✓			
RAC					✓
CoRDAGe				✓	
Wrangler					✓
Kalimucho		✓			
StarCCM		✓			
Codewan			✓		
Cloudlet		✓	✓		
ORYA		✓			
DeployWare		✓		✓	
ADME		✓			
TUNe				✓	
SmartFrog		✓		✓	

(a) Nature du domaine

	Limitée	Avancée
FROGi		
R-OSGi	pannes	
Soft. Dock	connectivité du réseau	
QUIET	disponibilité des ressources	
Disnix		pannes, connexions, déconnexions
RAC		
CoRDAGe		qualité et disponibilité des ressources
Wrangler		dynamique du cloud
Kalimucho		connexions, déconnexions, QoS
StarCCM		contexte
Codewan		déconnexions, fragmentation du réseau
Cloudlet	mobilité *(manuelle)*	
ORYA		
DeployWare		
ADME		pannes d'hôte et perturbations
TUNe		dynamique de la grille
SmartFrog		pannes, disponibilité des ressources

(b) Gestion de la dynamique

Tableau 3.2 – Domaine de déploiement

3.4.3 Conception et expressivité

Nous avons présenté dans la section 2.2.1 les différents rôles joués par les parties prenantes. Ici, nous nous concentrons sur l'expressivité et le niveau d'abstraction dont les par-

ties prenantes peuvent bénéficier au moment de l'expression des propriétés de déploiement, et de l'expertise nécessaire qu'elles doivent avoir.

La table 3.3a montre ce qui doit être exprimé : soit le plan de déploiement (saisi statiquement), soit un ensemble de propriétés (contraintes, choix de conception, ou propriétés de configuration) à partir desquelles un plan de déploiement peut être calculé ou dynamiquement adapté (les deux approches peuvent aussi être combinées). L'expression de propriétés de déploiement se base sur du XML (Wrangler) ou un DSL (Software Dock, ADME), et dans certains cas, sur les styles de programmation à base de règles (StarCCM, ORYA). Plusieurs solutions (Disnix, CoRDAGe, DeployWare, TUNe) proposent l'utilisation de modèles et des transformations de modèles. Les modèles permettent un haut-niveau d'abstraction pour les concepteurs. Ils facilitent la séparation des besoins orientés rôle. Ils peuvent aussi supporter la validation des propriétés de déploiement et ainsi la fiabilité du déploiement peut être atteinte (Disnix, DeployWare). Un autre problème concerne la transparence du domaine lors de l'expression des caractéristiques du déploiement. Cette exigence est particulièrement importante dans le cas de domaines ouverts et instables, dans lesquels les machines peuvent ne pas être connues à la conception et découvertes dynamiquement. Dans ce cas, il n'est pas possible d'établir statiquement un plan de déploiement complet. La table 3.3b met en évidence que l'impact des solutions est très limité sur ce point.

	Plan	Propriétés	Configuration
FROGi	*n/a*	✓	
R-OSGi	✓		
Soft. Dock	*n/a*	✓	✓
QUIET	✓		✓
Disnix		✓	
RAC			✓
CoRDAGe		✓	
Wrangler		✓	
Kalimucho	✓	✓	✓
StarCCM		✓	✓
Codewan	*n/a*	*n/a*	*n/a*
Cloudlet	✓		
ORYA		✓	
DeployWare	✓	✓	✓
ADME		✓	
TUNe		✓	
SmartFrog	✓	✓	✓

(a) Nature de la spécification

	Transparence du domaine
FROGi	*n/a*
R-OSGi	
Soft. Dock	✓
QUIET	
Disnix	
RAC	✓(cloud)
CoRDAGe	
Wrangler	✓(partielle)
Kalimucho	✓
StarCCM	
Codewan	✓
Cloudlet	
ORYA	
DeployWare	
ADME	
TUNe	
SmartFrog	✓(partielle)

(b) Transparence du domaine

Tableau 3.3 – Expression du déploiement

La table 3.4 met en évidence l'expertise requise par les concepteurs du déploiement. Peu de technologies sont utilisables par des personnes inexpérimentées, et pour la plupart, elles sont utilisées par des spécialistes, qui jouent à la fois le rôle de concepteur du déploiement et d'administrateur du système (elles supposent qu'ils contrôlent les ressources).

	Expertise du concepteur
FROGi	Fractal ADL, OSGi
R-OSGi	OSGi
Soft. Dock	déploiement, langage DSD
QUIET	déploiement
Disnix	administration du système
RAC	configuration et production de VA
CoRDAGe	*faible*
Wrangler	déploiement, virtualisation, administration
Kalimucho	système Kalimucho
StarCCM	déploiement, adaptation
Codewan	dépôt
Cloudlet	*faible*
ORYA	stratégies et activités de déploiement
DeployWare	*en accord avec l'expertise des parties prenantes*
ADME	déploiement et gestion de l'application
TUNe	déploiement et conception de modèles
SmartFrog	configuration, gestion de cycle de vie

Tableau 3.4 – Expertise du concepteur du déploiement

3.4.4 Réalisation du déploiement

Les objectifs des solutions en terme d'activités de déploiement, d'architecture du système de déploiement et de décentralisation du contrôle, et du bootstrap du système de déploiement sont résumés ici.

Activités de déploiement La table 3.5 montre quelles activités sont gérées par les différentes solutions. Les abréviations Dép., Inst., Act., MàJ., Adapt., Reconf., Redist., Desac., Desins., and Ret. font référence respectivement aux activités de dépôt, installation, activation, mise à jour, adaptation, reconfiguration, redistribution, désactivation, désinstallation et retrait. Il est à noter que l'installation et l'activation (et d'une manière étendue, désactivation et désinstallation) sont généralement gérées.

Les problèmes liés à la dynamique sont gérés de différentes manières, ou pas pris en compte. Plusieurs solutions visent les modifications dynamiques du plan de déploiement (Disnix, Kalimucho, StarCCM, ADME), dans certains cas au moyen de techniques d'allocations de ressources (CoRDAGe, Wrangler). Les autres solutions visent la reconfiguration ou le transfert de calcul (RAC, Cloudlet, TUNe, SmartFrog) ou la distribution de composants (Codewan).

	Dép.	Inst.	Act.	MàJ.	Adapt.	Reconf.	Redist.	Desac.	Desins.	Ret.
FROGi		✓	✓	✓		✓		✓	✓	
R-OSGi		✓	✓	✓				✓	✓	
Soft. Dock	✓	✓	✓	✓	✓	✓		✓	✓	✓
QUIET	✓	✓							✓	
Disnix	✓	✓	✓	✓		✓	✓	✓	✓	
RAC	✓	✓	✓			✓				
CoRDAGe		✓	✓			✓	✓	✓	✓	
Wrangler		✓				✓	✓			
Kalimucho		✓		✓	✓	✓	✓		✓	
StarCCM					✓	✓	✓			
Codewan	✓	✓								
Cloudlet		✓	✓					✓	✓	
ORYA		✓		✓						
DeployWare		✓	✓					✓	✓	
ADME		✓	✓			✓	✓			
TUNe		✓	✓			✓				
SmartFrog		✓	✓			✓		✓	✓	

Tableau 3.5 – Activités de déploiement couvertes

Contrôle et architecture Plusieurs solutions se concentrent sur l'architecture distribuée du système qui supporte le déploiement. Elles visent la décentralisation (Software Dock, QUIET, Wrangler, Kalimucho, Codewan), la sensibilité au contexte (StarCCM), l'auto-adaptation et l'autonomie (Disnix, ADME).

La table 3.6 indique comment le processus de déploiement est contrôlé – centralisé ou décentralisé – et montre que certaines solutions sont complètement décentralisées (Software Dock, QUIET, Kalimucho, Codewan, DeployWare, TUNe, SmartFrog). Il convient de noter qu'ORYA est dédié à l'expression des stratégies de déploiement afin de supporter la spécification du processus de déploiement.

	Centralisé	Décentralisé
FROGi	✓	
R-OSGi	✓	
Soft. Dock		✓
QUIET		✓
Disnix	✓	
RAC	✓	✓
CoRDAGe	✓	
Wrangler	✓	✓
Kalimucho		✓
StarCCM	✓	
Codewan		✓
Cloudlet	✓	✓
ORYA	✓	
DeployWare		✓
ADME	✓	✓
TUNe		✓
SmartFrog		✓

Tableau 3.6 – Contrôle du déploiement

Bootstrap En pratique, les systèmes de déploiement incluent des bootstrap – la plupart des solutions supposent la disponibilité du bootstrap sur les machines sans envisager la question précisément. La table 3.7 montre la nature de ce bootstrap : spécifique, non spécifique (spécifique à une technologie qui est largement diffusée), ou standard. Plusieurs solutions se basent sur un bootstrap spécifique (c'est-à-dire développé pour le système de déploiement). Certaines se basent sur des outils ou des plateformes existants, mais peu d'entre eux (RAC, Wrangler, SmartFrog) sont réellement indépendants de la technologie.

	Spécifique	Non spécifique	Standard
FROGi	FrogiBundleActivator	plateforme OSGi	
R-OSGi		plateforme OSGi	
Soft. Dock	field dock	plateforme Voyager	
QUIET		plateforme JADE	
Disnix	DisNixService		
RAC			plateforme de virtualisation
CoRDAGe		middleware de grille, ADAGE	
Wrangler			plateforme de virtualisation
Kalimucho	plateforme Kalimucho		
StarCCM	middleware StarCCM		
Codewan	plateforme Codewan		
Cloudlet	baseVM et KCM		
ORYA	*n/a*	*n/a*	*n/a*
DeployWare	DeployWare runtime		
ADME	infrastructure Cingal		
TUNe	TUNe runtime		
SmartFrog			Java, ftp, ssh

Tableau 3.7 – Nature du bootstrap

Contribution

L'état de l'art en matière de déploiement automatique est analysé dans le cadre de quatre questions (déployer quoi ? déployer où ? comment concevoir ? comment réaliser le déploiement ?) déclinées en sept critères. L'analyse montre que la dynamique et l'extensibilité du système logiciel sont rarement prises en compte. Concernant le domaine, les solutions proposées visent différents types d'appareils. Mais, à une exception près, elles ne considèrent pas à la fois l'hétérogénéité, l'extensibilité et la dynamique. Il existe des solutions décentralisées mais les évolutions dynamiques du plan de déploiement sont généralement limitées. Pour concevoir et exprimer le plan, certaines solutions proposent l'utilisation de modèles ou de langages dédiés, en particulier de DSL. Cependant, le besoin de transparence du domaine lors de la conception, particulièrement important pour les systèmes répartis multi-échelles, n'est pas satisfait. Ainsi, de manière générale, les solutions de déploiement existantes ne répondent que de manière incomplète ou inefficace aux exigences du déploiement des systèmes répartis multi-échelles.

4 Processus de déploiement

Problématique

Alors que les processus de déploiement traditionnels sont le plus souvent centralisés et dirigés par un opérateur humain, le processus de déploiement des systèmes répartis multi-échelles doit être décentralisé, automatisé et sensible au contexte. Le rôle d'opérateur de déploiement doit y être joué par une application de niveau middleware : le système de déploiement.

4.1 Introduction

La norme [ISO 9000 :2005, 2009] définit un processus comme un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. La conception, puis la réalisation et éventuellement le contrôle du déploiement sont les activités principales du processus de déploiement qui prend en entrée les éléments à déployer et leurs caractéristiques, et permet en sortie d'obtenir une application disponible à l'utilisation, selon la définition de Carzaniga *et al*.

Les processus de déploiement usuels sont centralisés et dirigés par un opérateur humain. Ils sont adaptés au déploiement d'applications dans un environnement hiérarchique, tels un réseau d'entreprise ou une grappe de serveurs. Ils sont généralement pilotés depuis une unité centrale où un opérateur humain cumule les rôles de concepteur et d'opérateur du déploiement.

Dans la section 2.3, nous avons montré qu'une solution de déploiement pour les systèmes répartis multi-échelles devait organiser et automatiser un certain nombre d'opérations dans le cadre d'un processus en partie décentralisé et sensible au contexte. Ce processus doit permettre la prise en compte de la dynamique du domaine de déploiement ainsi que celle du système logiciel, et permettre une séparation claire entre les rôles. En particulier, celui de concepteur doit être dévolu à un opérateur humain alors que le rôle d'opérateur peut être pris par le système de déploiement. Pour cela, il nous semble pertinent de repenser et définir un processus de déploiement adéquat.

Ce chapitre est organisé comme suit. Nous présentons une vue globale du processus de déploiement dans la section 4.2, que nous détaillons dans section 4.3. Dans les sections 4.3.1

et 4.3.2, nous présentons la phase de préparation des appareils du domaine de déploiement et de l'application. Ensuite, dans la section 4.3.3, nous présentons le déploiement initial (expression des propriétés, génération du plan et réalisation du plan) dont l'objectif est de rendre le logiciel utilisable. Dans la section 4.3.4, nous ciblons la partie dynamique du déploiement (supervision et opérations sur les composants à l'exécution), dont l'objectif est le maintien du logiciel en état opérationnel en prenant en compte la dynamique du domaine et de l'application. Enfin, dans les sections 4.3.5 et 4.3.6 nous traitons les activités post-exécution et de retrait de l'application. Pour chaque point, la problématique et une analyse en relation avec l'état de l'art sont données, permettant une justification de la proposition qui s'ensuit.

4.2 Vue globale du processus

Le processus que nous proposons organise les différentes activités de déploiement de la manière suivante. Dans un premier temps, le concepteur exprime une spécification. Puis, un plan de déploiement initial de l'application est généré automatiquement en fonction de la spécification et de l'état initial du domaine. Ensuite, le plan de déploiement initial est réalisé, puis révisé à l'exécution afin d'être adapté à la dynamique de l'application et à celle du domaine, ceci pour maintenir l'application en condition opérationnelle : cette partie du processus est appelée « déploiement dynamique ».

Dans ce processus, le rôle d'opérateur du déploiement est joué par une application de niveau middleware : le système de déploiement. C'est ce système de déploiement automatique (et autonomique) qui réalise les différentes opérations de déploiement.

Cette approche suppose une acquisition de l'état du domaine, initialement pour produire les données nécessaires à la génération du plan en fonction des appareils présents et des ressources disponibles, puis dynamiquement pendant l'exécution de l'application.

Pour décrire le processus, nous utilisons une notation graphique inspirée de SPEM [SPE]. La légende des diagrammes est donnée dans la figure 4.1. Trois types d'actions seront décrites (différenciées par leurs couleurs) : (i) les actions centralisées, qui sont pilotées par une unité centrale, et s'effectuent sur un seul appareil, (ii) les actions mixtes, pilotées par une unité centrale mais dont l'action est réalisée sur l'ensemble ou une partie du domaine de déploiement, (iii) les actions décentralisées qui sont effectuées sur l'ensemble ou une partie du domaine de déploiement, sans initiative centrale. Sauf indication contraire, les actions décentralisées sont initiées par le système de déploiement.

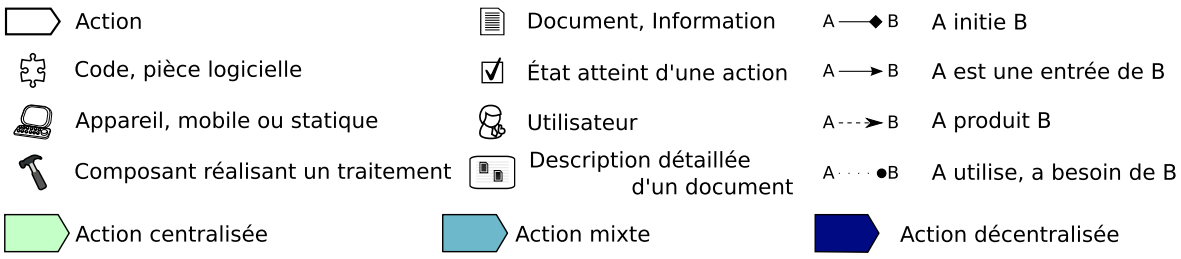


Figure 4.1 – Légende des diagrammes de processus

La figure 4.2 donne une vue globale du processus de déploiement.

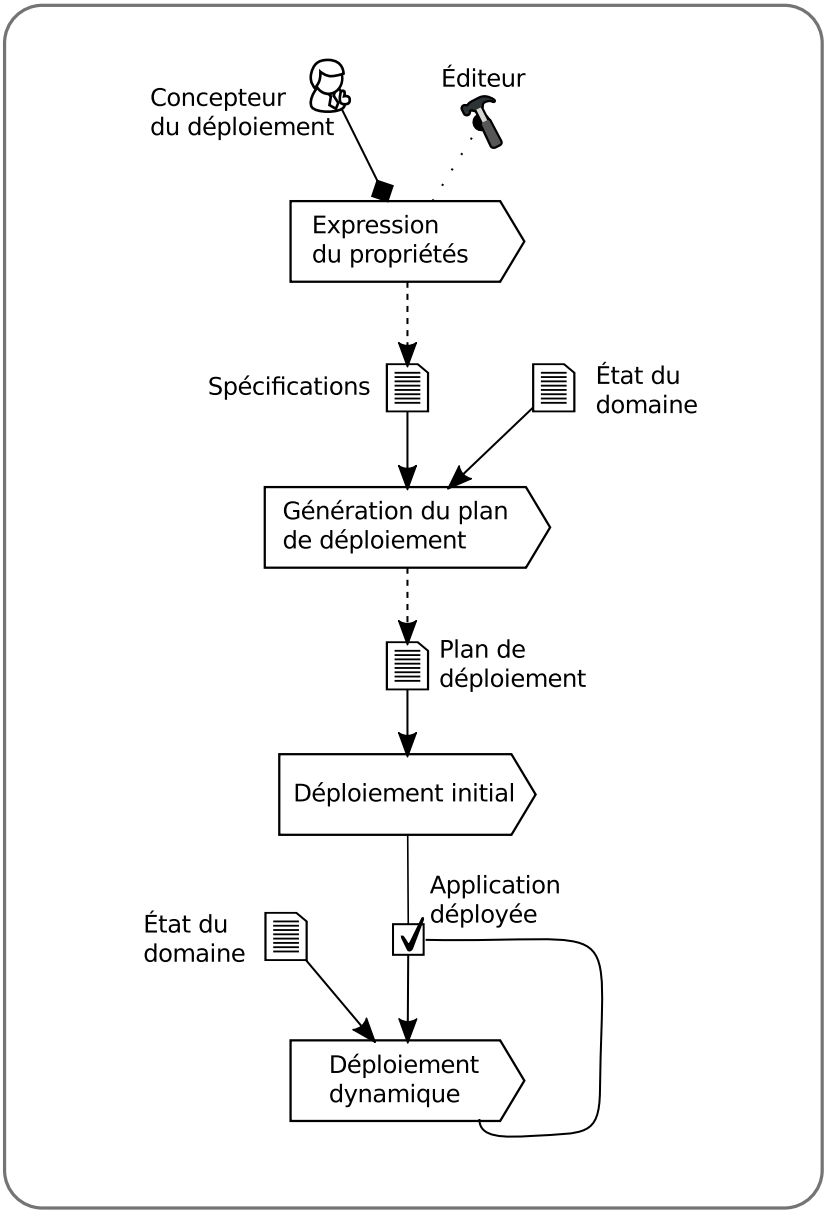


Figure 4.2 – Vue globale du processus de déploiement

4.3 Le processus en détail

Dans cette section, nous raffinons le processus présenté dans la section précédente et nous entrons dans le détail des différentes activités.

Ce processus mobilise deux composants majeurs :
- Le langage dédié et son éditeur qui permettent au concepteur du déploiement d'exprimer les propriétés du déploiement.
- Le middleware de déploiement, réparti sur les appareils du domaine, qui comprend
  - un gestionnaire de domaine,
  - un moniteur de déploiement, centralisé, en charge de la génération et réalisation du déploiement initial,
  - sur chaque appareil du domaine, un support local de déploiement qui réalise les

opérations locales de déploiement.

Le domaine de déploiement étant dynamique, il n'est pas connu au moment de la conception. Le gestionnaire de domaine est une entité centralisée qui permet d'enregistrer les entrées et les sorties des appareils dans le domaine de déploiement. En permanence, le gestionnaire du domaine a une connaissance de la composition du domaine de déploiement.

Le moniteur de déploiement est une entité centralisée du système de déploiement responsable du déploiement initial. C'est sur l'appareil qui héberge le moniteur que les informations (spécification, état du domaine) sont traitées et que la génération du plan de déploiement est effectuée, puis sa réalisation pilotée. Cette génération demande une certaine puissance de calcul.

4.3.1 Préparation des appareils

Une partie du système de déploiement doit être présent sur l'ensemble des appareils du domaine sous la forme d'un support local de déploiement. L'exécution de cette partie du système permet également d'enregistrer un appareil dans le domaine de déploiement, ainsi que de gérer les problèmes d'administration, en permettant par exemple une interaction avec le propriétaire de l'appareil pour obtenir des droits d'accès particuliers. Nous appelons cette partie du système de déploiement le **bootstrap**.

La préparation des appareils est donc une étape préliminaire aux opérations de déploiement qui a pour objet d'installer et activer un bootstrap sur les différents appareils.

4.3.1.1 Installation et activation du bootstrap

Analyse Le bootstrap est un élément essentiel dans le cadre des systèmes multi-échelles vu le nombre d'appareils hétérogènes et la présence dans les appareils visés d'appareils mobiles dont l'administrateur est le propriétaire, utilisateur du logiciel. La présence de bootstrap et sa facilité d'installation a un impact sur l'adoption du système de déploiement, et donc du nombre potentiel d'appareils composant le domaine de déploiement.

Plusieurs travaux de déploiement font l'hypothèse qu'un bootstrap (ou une forme de plateforme d'exécution minimale) est déjà installé sur l'appareil hôte, sans préciser la manière dont il est installé. Il s'agit pour les plus simples d'un service ssh permettant d'exécuter des commandes sur un appareil distant, ou encore d'un système local de déploiement *ad hoc* exécuté en tant que service et supposé lancé par l'utilisateur au démarrage de l'appareil.

Proposition La figure 4.3 décrit le processus d'installation du bootstrap de déploiement. Sur son appareil, l'utilisateur initie l'installation et l'activation du bootstrap. Au final, le bootstrap est bien installé et actif sur l'appareil de l'utilisateur.

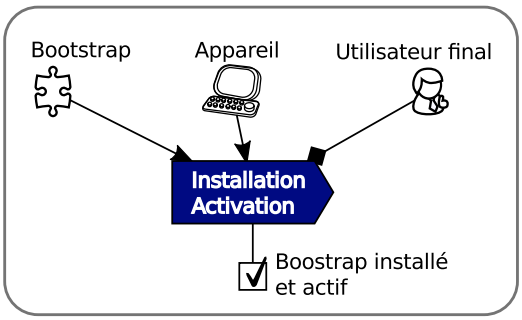


Figure 4.3 – Installation et activation du bootstrap

4.3.1.2 Enregistrement d'un appareil dans le domaine

Analyse Le système de déploiement doit être capable au début de l'exécution du déploiement de récupérer la liste de l'ensemble des appareils dans le domaine de déploiement.

Dans les travaux existants, à l'exception de Codewan qui est spécifique aux réseaux MANETs et qui permet la découverte des appareils, le concepteur du déploiement énumère les appareils appartenant au domaine de déploiement, ou bien l'opérateur du déploiement effectue l'enregistrement de l'appareil une fois le système de déploiement lancé. Cette action effectuée par l'opérateur humain peut rapidement gagner en complexité dans le cadre d'un grand nombre d'appareils.

Proposition La figure 4.4 décrit l'enregistrement automatique d'un appareil dans le domaine de déploiement. L'appareil, ayant un bootstrap qui s'exécute, s'enregistre auprès d'un gestionnaire du domaine de déploiement. L'ensemble des appareils qui s'enregistrent sur ce serveur constitue le domaine de déploiement.

Ainsi, ni le concepteur, ni l'opérateur de déploiement n'a d'actions à effectuer afin d'ajouter un appareil dans le domaine de déploiement.

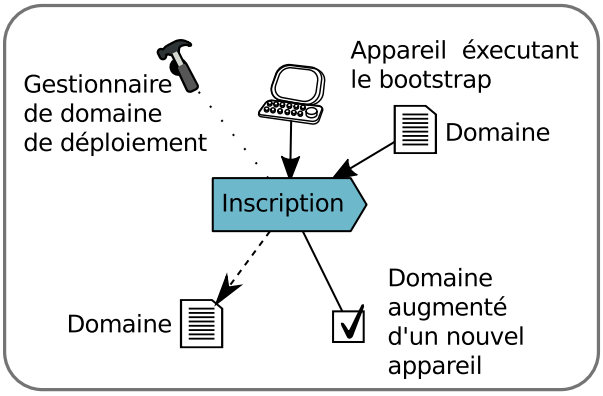


Figure 4.4 – Enregistrement d'un appareil dans le domaine

4.3.2 Préparation de l'application

4.3.2.1 Dépôt (mise à disposition) des composants

Analyse Dans le cadre des systèmes visées l'unité de déploiement est le composant. Ainsi, le producteur doit fournir son application sous forme de composants logiciels conformes au modèle choisi pour être exploitables par le système de déploiement. Généralement, il s'agit d'un modèle de composant unique.

La majorité des travaux s'intéressent au déploiement d'application à base de composants. Seules FROGi, Software Dock, QUIET, Cloudlet et ORYA prévoient le déploiement d'applications monolithiques.

Proposition Dans notre processus, nous supposons que le concepteur du déploiement a développé ou encapsulé ses composants dans le modèle de composants de référence du système de déploiement. Le composant est ainsi disponible au déploiement de par le système de déploiement sans aucune autre opération supplémentaire.

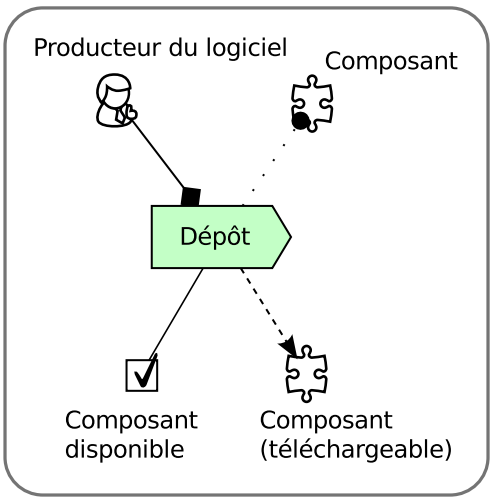


Figure 4.5 – Mise à disposition d'un composant

La figure 4.5 décrit la mise à disposition d'un composant, par son producteur, sur un dépôt de composants logiciels.

4.3.3 Déploiement initial

Le déploiement initial a pour objet de construire puis de mettre en œuvre un premier plan de déploiement conforme aux spécifications exprimées par le concepteur et dépendant du domaine de déploiement. Ici, la mise en œuvre consiste à installer et à activer les composants du système logiciel à déployer.

4.3.3.1 Production du plan de déploiement

Analyse La production du plan de déploiement est composées de différentes actions ordonnées. Chaque action produit un résultat qui sera pris en entrée de la suivante, afin

d'aboutir à un plan de déploiement et un ensemble de propriétés.

Dans les travaux existants, majoritairement, un plan de déploiement statique doit être décrit par le concepteur du déploiement. Ce plan de déploiement peut être agrémenté par des propriétés de vérification et de validation (Disnix, DeployWare). ADME permet une description du déploiement basée sur l'expression de contraintes de déploiement. Ce choix d'expression permet de relever l'humain de la tâche complexe d'attribution des correspondances. De plus, pour faciliter l'expression du déploiement certains travaux proposent une abstraction lors l'expression du déploiement, en utilisant des langages (Software Dock, Wrangler, etc.) ou des modèles (Disnix, TUNe, etc.).

Proposition La figure 4.6 décrit le processus de production du plan de déploiement. Tout d'abord, le concepteur du déploiement spécifie les propriétés de déploiement en utilisant un langage dédié.

Des collecteurs d'information sont présents dans le bootstrap pour la récupération d'informations utiles à la supervision du système. Ces collecteurs d'information sont appelés des **sondes**. Afin de pouvoir contrôler la satisfaction des propriétés du déploiement, le concepteur du déploiement doit aussi spécifier les sondes à utiliser pour récupérer les informations pertinentes sur le domaine. Ces spécifications se font aussi en utilisant le langage de haut-niveau. La liste des sondes nécessaires au déploiement ainsi que la localisation de leur code sont extraites de la spécification du déploiement. Les codes des sondes sont ensuite disséminés : ils sont téléchargés et distribués sur le domaine de déploiement, puis installés et activés. Le système de déploiement effectue alors un sondage local, et renvoie les résultats de ce sondage (l'ensemble de ces informations constitue l'état du domaine) au moniteur de déploiement.

Les propriétés sont donc à la suite d'une d'analyse, transformées en contraintes (la formalisation de cette transformation est décrite à la section 6.3). À partir de ces contraintes et de l'état du domaine, et au moyen d'un solveur de contraintes, un plan de déploiement est produit. Ce plan est celui qui sera mis en œuvre initialement – il évoluera ensuite dynamiquement en fonction de l'état du domaine, *via* le système de déploiement autonome capable d'adaptation proactive aux changements de contexte.

L'action de génération du plan de déploiement produit aussi deux ensembles de propriétés : les propriétés à vérifier lors de l'installation, et les propriétés à préserver à l'exécution. Lors de l'expression des propriétés, le concepteur spécifie quelles sont les propriétés initiales, et celles à préserver dynamiquement.

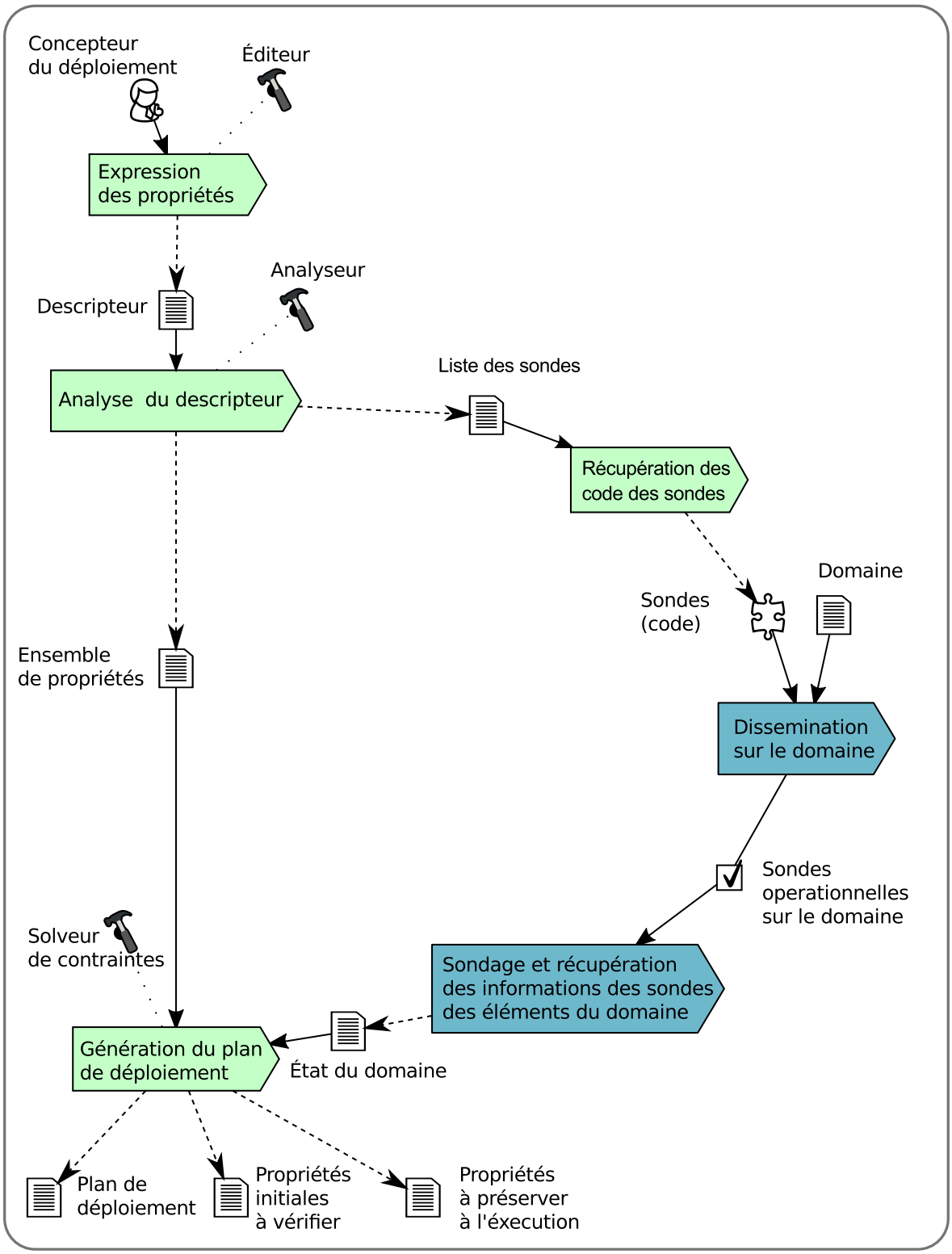


Figure 4.6 – Production du plan de déploiement

4.3.3.2 Installation du système logiciel multi-échelle

Analyse Le plan de déploiement a été généré en fonction des spécifications du concepteur et de l'état du domaine. Ce plan contient toutes les informations nécessaires à l'installation des composants sur les différents appareils. L'état du domaine peut avoir évolué depuis la récupération de l'état du domaine lors de la production du plan de déploiement. Ainsi, le système de déploiement doit effectuer cette installation en adéquation avec le plan et s'assurer que les propriétés initiales sont toujours satisfaites.

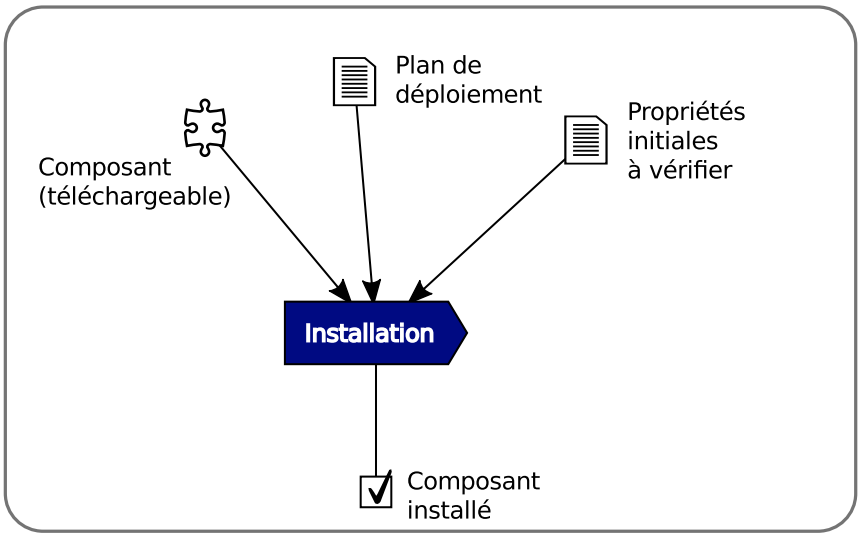


Figure 4.7 – Installation d'un composant

Proposition La figure 4.7 décrit l'installation sur un appareil d'un composant du système multi-échelle. Le plan de déploiement ainsi que les propriétés initiales étant connus, le système de déploiement procède à l'installation localement. Il télécharge le composant à installer sur l'appareil et vérifie que les propriétés initiales (celles qui ont été exprimées lors de la description du déploiement) sont toujours valides. Si c'est le cas, il installe le composant sur l'appareil.

Avec la vérification des propriétés de déploiement au moment de l'installation, le système de déploiement s'assure bien que les propriétés n'ont pas été violées, sinon, une boucle autonomique peut prendre la relève pour résoudre le problème, sans en référer directement à l'opérateur de déploiement.

4.3.3.3 Activation des composants

Analyse À l'installation, toutes les propriétés ayant été vérifiées, l'activation du composant peut s'effectuer.

Proposition La figure 4.8 décrit l'activation d'un composant sur un appareil. Le composant à activer est déjà installé sur l'appareil. Le système de déploiement active le composant. Un ensemble de propriétés est aussi nécessaire pour la vérification de la satisfaction des propriétés relatives au composant à l'exécution. Ces propriétés diffèrent des propriétés initiales, les sondes utilisées pour la récupération des informations nécessaires aux propriétés

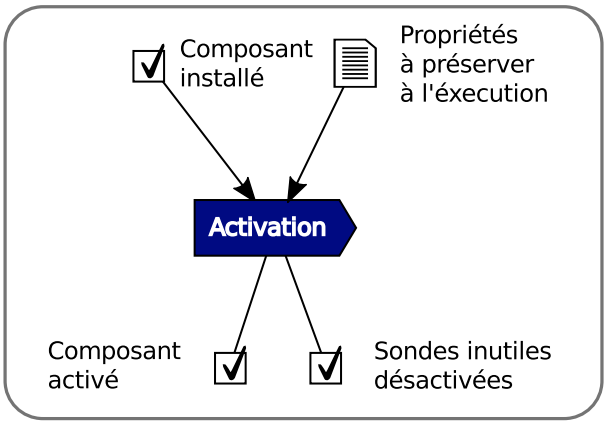


Figure 4.8 – Activation d'un composant

initiales sont désactivées. Les sondes relatives aux propriétés des composants qui ne sont pas déployés sur cet appareil sont aussi désactivées.

4.3.3.4 Vue globale de la phase initiale de déploiement

La figure 4.9 donne une vue globale du processus de déploiement initial.

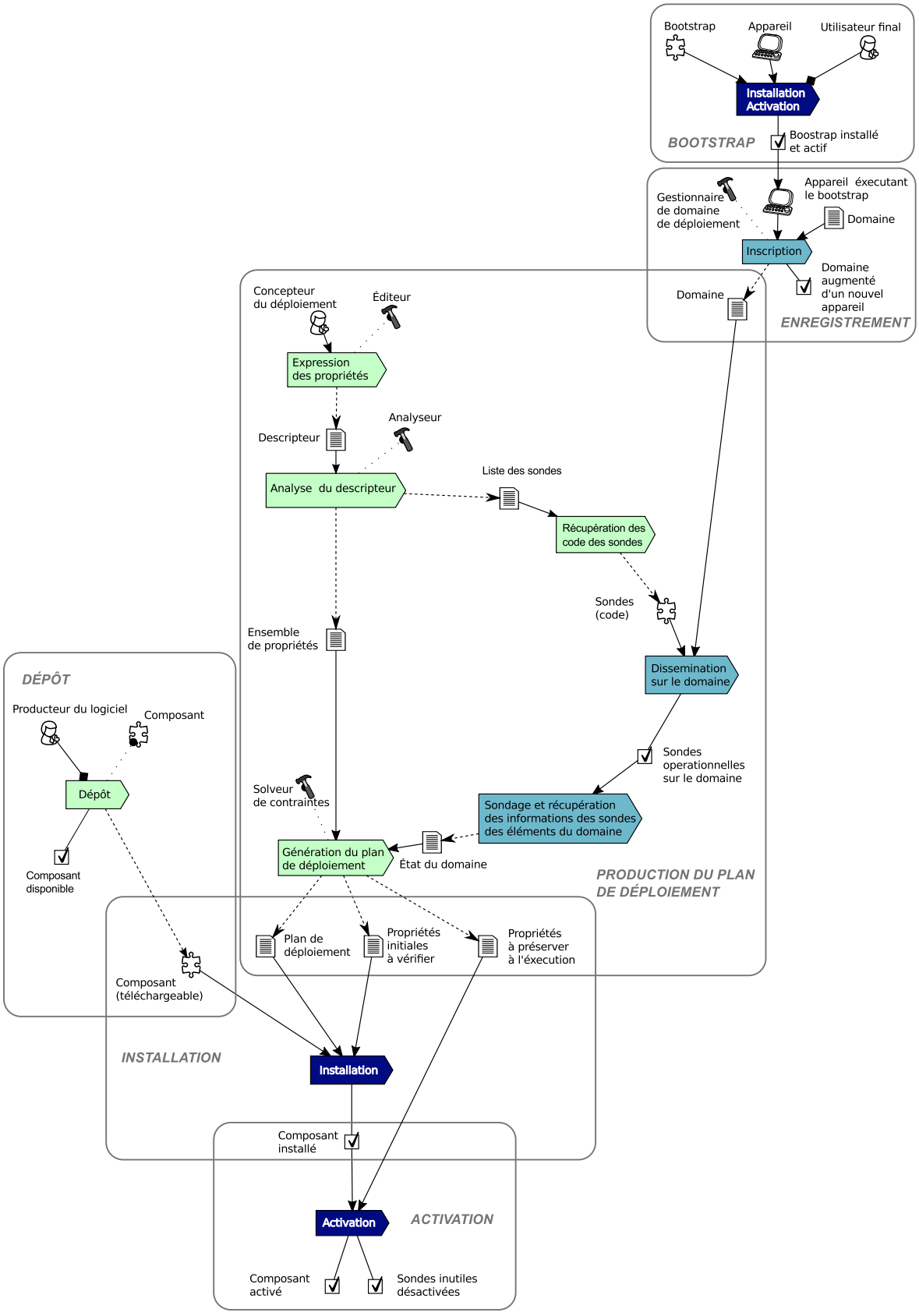


Figure 4.9 – Vue globale du processus de déploiement initial

4.3.4 Déploiement dynamique

Le déploiement dynamique a pour objet d'adapter le plan de déploiement de l'application (pendant son exécution) afin de maintenir celle-ci dans un état opérationnel, tout en respectant les propriétés exprimées par le concepteur. Le déploiement dynamique prend en compte le déploiement continu (dynamique du domaine), de la réaction aux variations du domaine, et du déploiement incrémental (dynamique de l'application).

4.3.4.1 Déploiement continu

Analyse La dynamique du domaine de déploiement est un des point essentiels dans le cadre de systèmes répartis multi-échelles. Lorsque le plan de déploiement est spécifié par le concepteur du déploiement, dans les travaux étudiés il est immuable, sauf par action de l'opérateur du déploiement humain. Cette action de l'opérateur est problématique dans un contexte de grand nombre d'appareils et de composants. Si à chaque fois que le domaine évolue, un solveur de contraintes centralisé doit être relancé (comme pour ADME), les réponses successives à ces changements d'état demanderaient beaucoup de temps. De plus, une fois le nouveau plan généré, l'état du domaine aura encore évolué, ce qui nécessitera une regénération du plan de déploiement. Le système de déploiement doit pouvoir prendre en compte, sans action de l'opérateur, les appareils qui entrent dans le domaine de déploiement, et y installer les composants concernés, si besoin.

Les travaux prenant en compte cette dynamique requièrent une action de la part de l'opérateur de déploiement (Kalimucho, Cloudlet, etc.). Seul Codewan permet la gestion autonomique de la dynamique de l'environnement, dû à la nature du réseau visé (MANET).

Proposition Lorsqu'un nouvel appareil (ayant un bootstrap) entre dans le domaine de déploiement et si le concepteur a exprimé dans sa spécification des propriétés dynamiques sur un ou plusieurs composants, ce ou ces composants sont installés et activés sur cet appareil. Les **propriétés dynamiques** sont des exigences spécifiques à la prise en compte de nouveaux appareils entrant ou sortant du domaine de déploiement. Le concepteur de déploiement peut spécifier qu'un composant doit être déployé sur chaque appareil entrant (à condition que les autres propriétés du composant soient valides).

La figure 4.10 décrit le déploiement continu. Un appareil apparaît dans le domaine de déploiement. Le support local du système de déploiement vérifie si l'appareil est compatible avec une propriété dynamique d'un composant (c'est-à-dire qu'il satisfait bien l'ensemble des propriétés du composant ayant une propriété dynamique). S'il est compatible, le composant adéquat est installé et activé.

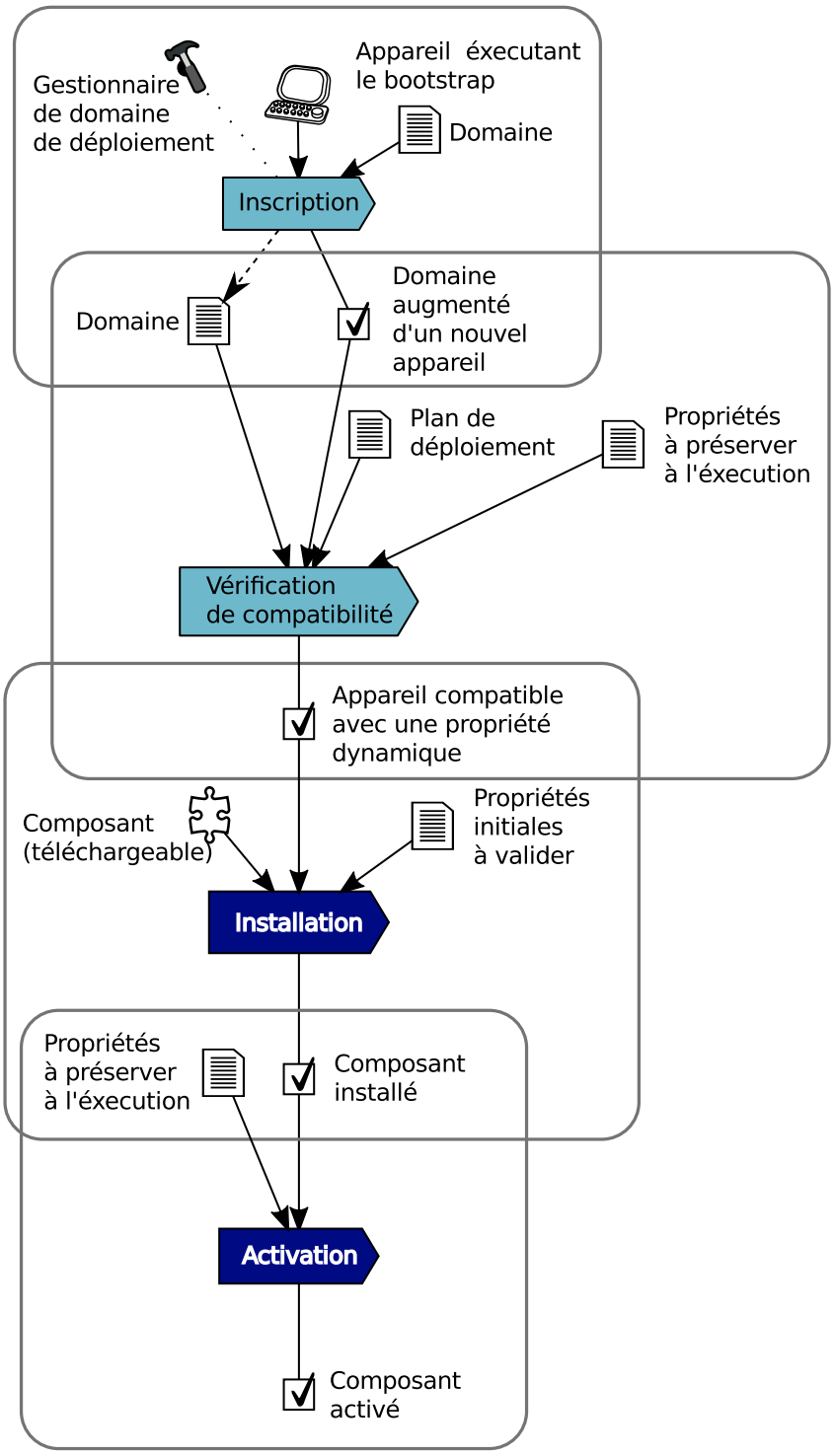


Figure 4.10 – Déploiement continu

4.3.4.2 Supervision et analyse

Cette partie décrit le maintien en condition opérationnelle du système déployé à l'exécution en cas de changement dans l'environnement d'exécution. Le système de déploiement doit pouvoir percevoir les changements de l'environnement d'exécution, et

s'y adapter dynamiquement afin que le système déployé retrouve un état opérationnel, c'est-à-dire respecter les spécifications. Nous nous basons pour cela sur les principes de l'informatique autonomique. Les principes de l'informatique autonomique sont présentés dans le chapitre 7, ainsi que leur apport au déploiement. Nous décrivons ici la supervision du système, l'analyse et prise de décision. Ceci correspond, pour le système de déploiement, à la partie *Monitor-Analyze-Plan* -MAP- de la boucle autonomique de Kephart et Chess [Kephart and Chess, 2003].

Perception d'un changement dans l'environnement

Analyse Les **perturbations** sont des évènements susceptibles de remettre en cause le plan de déploiement. Les perturbations sont le propre des environnements dynamiques. Elles induisent une inconsistance dans le plan de déploiement qui est normale dans le cadre de déploiement autonomique. Le système de déploiement doit pouvoir percevoir ces perturbations.

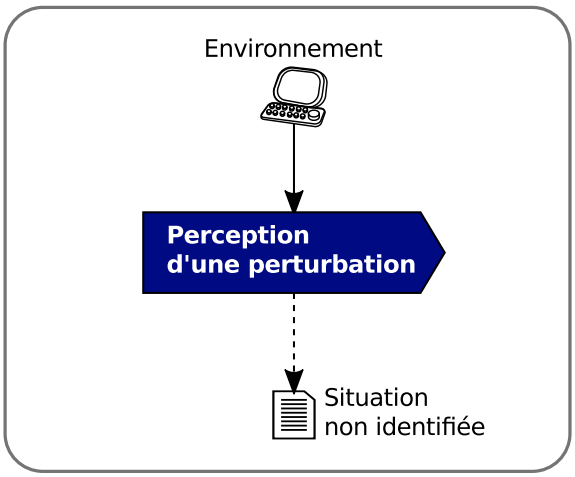


Figure 4.11 – Perception d'un changement dans l'environnement

Proposition La figure 4.11 décrit le processus de perception d'une perturbation. Si le système de déploiement perçoit localement un changement de l'état de l'appareil (par ex. la déconnexion d'un périphérique faisant partie des périphériques à surveiller par les sondes), une situation d'adaptation qui reste à identifier est produite dans le but de pouvoir s'y adapter.

Identification d'une situation de changement dans l'environnement

Analyse Le système de déploiement doit pouvoir analyser la perturbation perçue et l'analyser afin d'identifier la situation d'adaptation rencontrée.

Proposition La figure 4.12 décrit le processus d'identification d'une situation. Le système de déploiement analyse une situation donnée et l'identifie. Cette identification peut

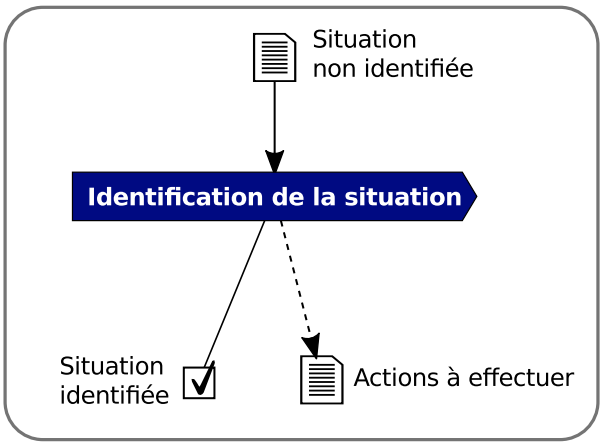


Figure 4.12 – Identification d'une situation de changement dans l'environnement

conduire soit à une adaptation d'un composant, avec ou sans arrêt de l'exécution de ce composant, soit à une reconfiguration, soit à une redistribution. L'identification permet aussi de déterminer quelles sont les actions à exécuter afin de s'adapter à ce changement dans l'environnement. L'identification peut aussi conduire à « remonter l'information » au système multi-échelle qui est déployé.

4.3.4.3 Adaptation et Mise à jour

Cette partie décrit les activités de déploiement qui interviennent à l'exécution du système déployé. Ceci correspond, pour le système de déploiement, à la partie *Execution* -E- de la boucle autonomique de Kephart et Chess [Kephart and Chess, 2003]. Ces opérations ont pour objet d'adapter ou de mettre à jour des composants déjà déployés. Elles peuvent nécessiter la désactivation ou la désinstallation de ces composants.

Adaptation sans arrêt

Analyse Lorsque l'environnement du composant change, une adaptation du composant peut être nécessaire. Le système de déploiement doit pouvoir répondre à une perturbation en effectuant une adaptation sans arrêt du composant, si cela est possible. Les travaux existants ne permettent pas une adaptation sans arrêter le composant (Kalimucho, StarCCM, etc.). Certaines situations d'adaptations peuvent permettre une adaptation sans arrêt, en modifiant la configuration du composant.

Proposition La figure 4.13 décrit l'adaptation d'un composant sans arrêter ce dernier. Une telle situation a été identifiée et le système de déploiement agit en conséquence. Une adaptation d'un composant sans arrêt mène à un changement dans la configuration de ce composant. Au final, le composant est à nouveau actif et fonctionnel.

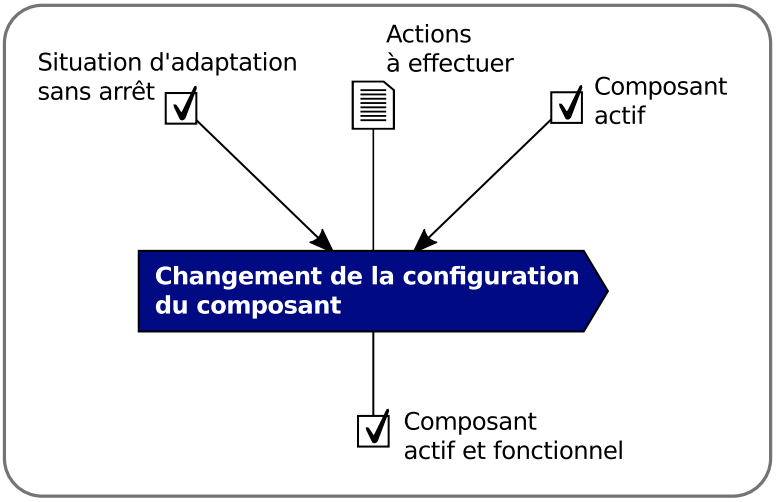


Figure 4.13 – Adaptation d'un composant sans arrêt

Adaptation avec arrêt

Analyse Une adaptation peut nécessiter l'arrêt du composant afin d'y effectuer des modifications de configuration. Le système de déploiement doit pouvoir répondre à une perturbation en effectuant une adaptation du composant, en l'arrêtant au préalable si nécessaire. Certains travaux prennent en compte l'adaptation de composant (Kalimucho, StarCCM) ou de l'application déployée (Software Dock).

Proposition La figure 4.14 décrit l'adaptation d'un composant avec arrêt. Comme pour l'adaptation sans arrêt, cette situation a été identifiée. Une adaptation d'un composant avec arrêt implique d'abord une désactivation du composant. Après la désactivation, les changements prévus sont effectués afin de procéder à l'adaptation du composant. Une fois ce travail terminé, le composant est à nouveau actif et fonctionnel.

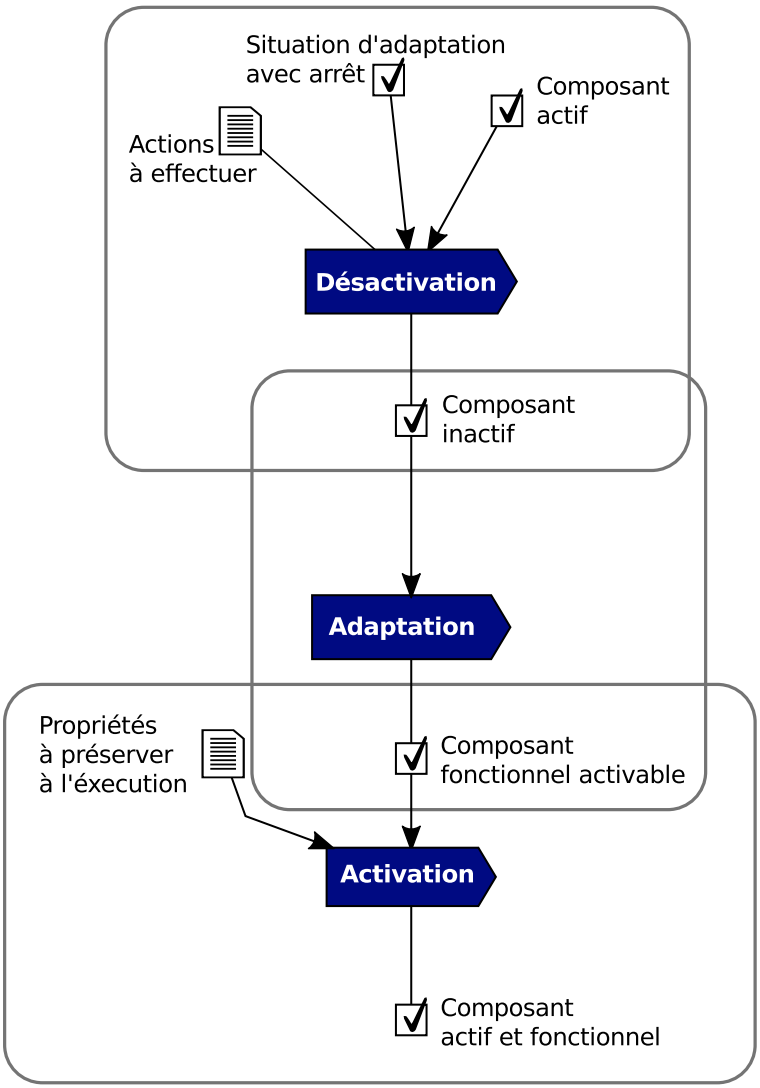


Figure 4.14 – Adaptation d'un composant avec arrêt

Mise à jour

Analyse Les différents composants du système multi-échelle évoluent avec le temps. Une mise à jour de certains composants est alors nécessaire. Le système de déploiement doit pouvoir prendre en compte cette évolution en effectuant une mise à jour du composant concerné. Cette activité est prise en compte par certains travaux, comme R-OSGi, Disnix, Kalimucho.

Proposition La figure 4.15 décrit le processus complet de mise à jour d'un composant. Lorsque le producteur de l'application met à disposition une nouvelle version d'un composant, l'information est disséminée à tous les appareils enregistrés dans le domaine de déploiement. Si un appareil possède ce composant, une mise à jour s'effectue (pour le moment, nous considérons que la mise à jour se fait automatiquement, à l'initiative du système de déploiement, mais elle pourrait être contrôlée si besoin par un administrateur ou un opérateur humain). La mise à jour s'effectue comme suit : tout d'abord, le nouveau compo-

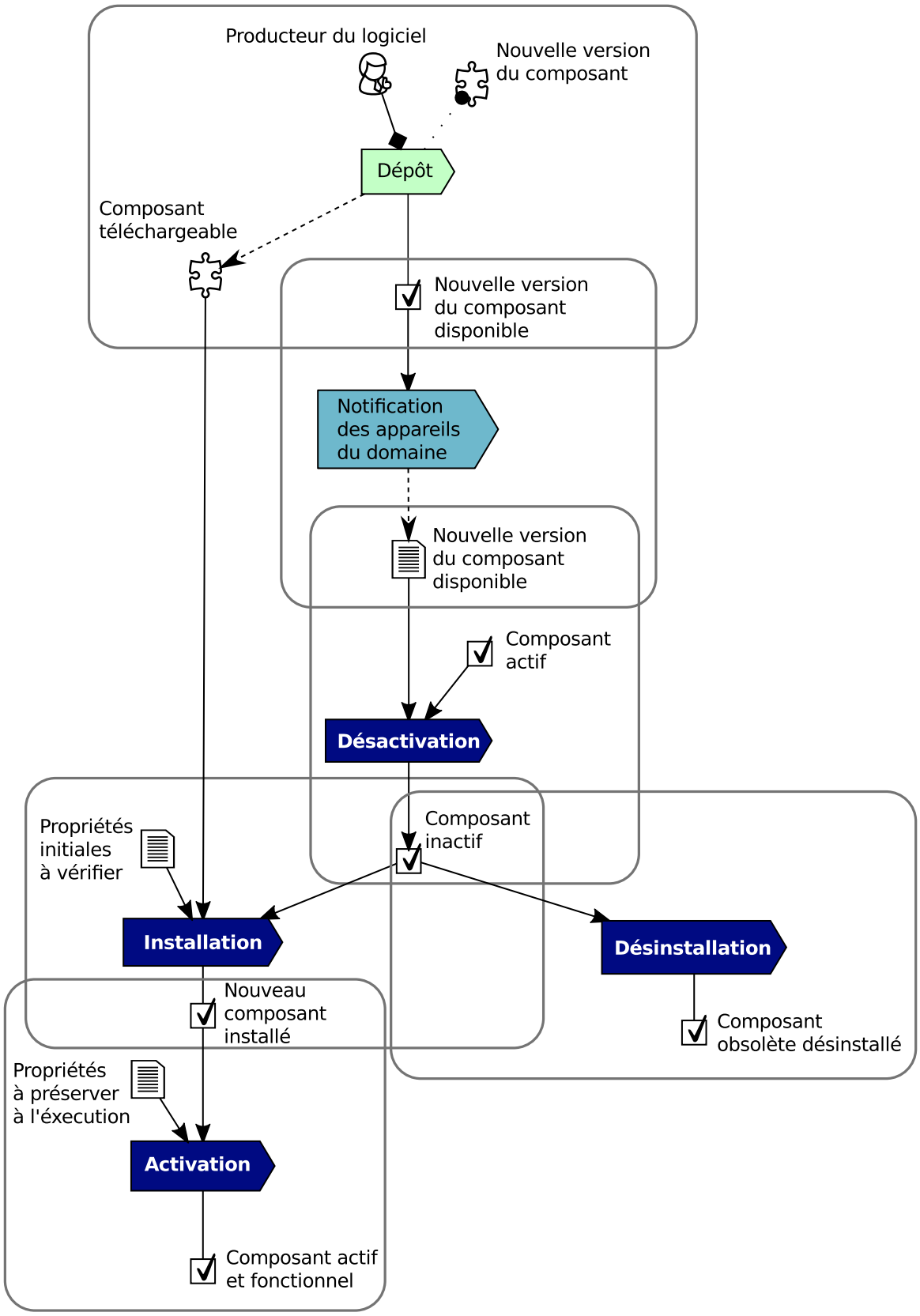


Figure 4.15 – Mise à jour d'un composant

sant est installé sur l'appareil et activé, puis le composant obsolète est désactivé et désinstallé de l'appareil. *In fine*, le composant est mis à jour, actif et fonctionnel.

4.3.4.4 Déploiement incrémental

Analyse Le système multi-échelle est amené à intégrer de nouveaux composants au système de composants déjà déployé. Cette évolution de l'application n'est généralement pas mise en avant dans les travaux existants (à part pour StarCCM par exemple). Il est nécessaire de pouvoir ajouter de nouvelles fonctionnalités au système déployé le système déployé sans en interrompre son fonctionnement.

Proposition Le déploiement incrémental a pour objet la prise en compte de la dynamique de l'application déployée, c'est-à-dire de ses nouveaux composants. Pour cela, le concepteur du déploiement décrit un ensemble de propriétés de déploiement spécifiques à ces nouveaux composants. Le déploiement de ces nouveaux composants peut dépendre de l'état des composants déjà déployés (le concepteur du déploiement exprime cette dépendance dans le descripteur). Ainsi, le déploiement incrémental peut imposer une adaptation de composants déjà déployés.

La figure 4.16 illustre le déploiement incrémental. L'opérateur du déploiement, humain dans ce cas, spécifie qu'un nouveau composant est à déployer. Puis, l'information est disséminée sur le domaine de déploiement. Enfin, le composant est installé et activé sur les différents appareils compatibles.

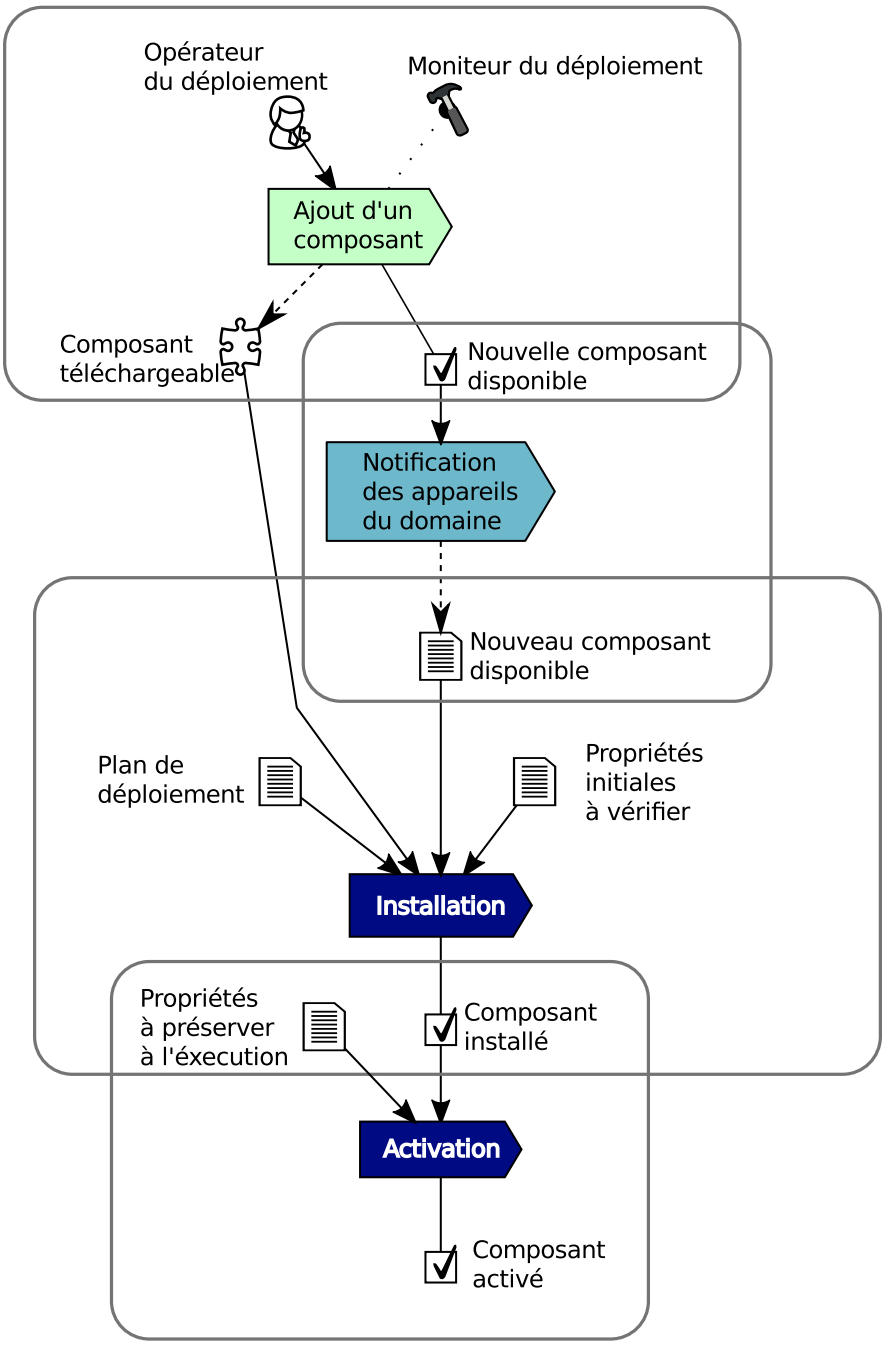


Figure 4.16 – Déploiement incrémental

4.3.5 Activités post-exécution

Les activités post-production peuvent provenir d'une demande d'arrêt générale du système déployé par l'opérateur du déploiement, ou bien de manière unitaire sur un composant à la demande d'une réaction à une situation d'adaptation.

4.3.5.1 Désactivation

Analyse Si le composant n'a plus besoin d'être actif sur un appareil, il est désactivé.

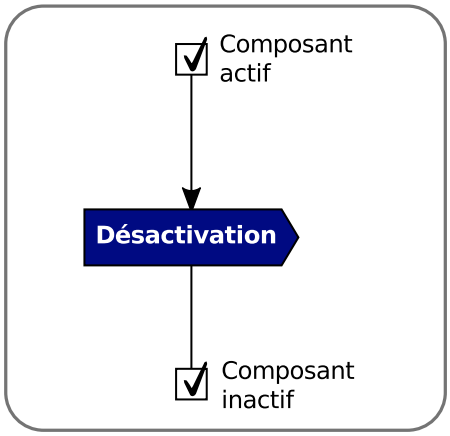


Figure 4.17 – Désactivation d'un composant

Proposition La figure 4.17 décrit la désactivation d'un composant. Lors d'une demande de désactivation, le système de déploiement désactive le composant.

4.3.5.2 Désinstallation

Analyse Une fois le composant désactivé et s'il n'est plus nécessaire (ne va pas faire l'objet d'une activation ultérieure), il peut être désinstallé de l'appareil.

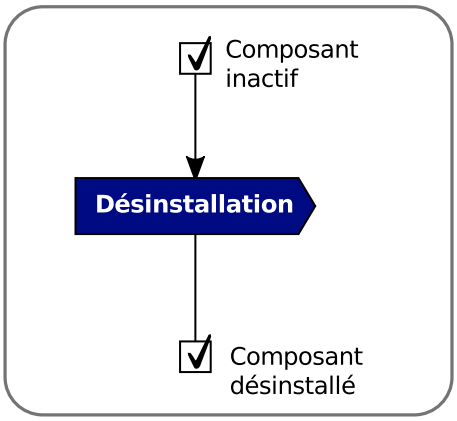


Figure 4.18 – Désinstallation d'un composant

Proposition La figure 4.18 décrit la désinstallation d'un composant. Une fois le composant désactivé, le système de déploiement désinstalle ce composant.

4.3.6 Retrait d'un composant

Analyse Certains composants deviennent obsolètes avec l'évolution de l'application. Ces composants n'étant plus utiles, ou néfastes (car producteurs d'erreur) à l'application, ils sont retirés du site producteur.

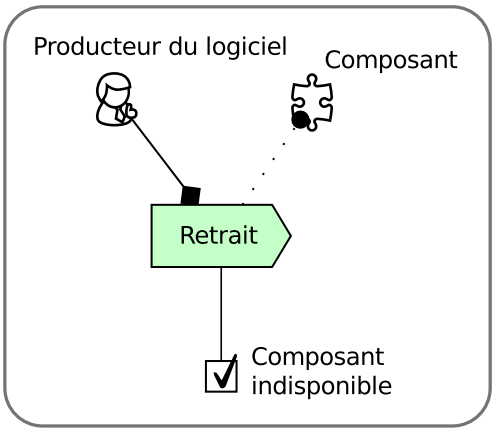


Figure 4.19 – Retrait d'un composant

Proposition La figure 4.19 décrit le retrait d'un composant par le producteur du déploiement sur le site producteur. Le composant obsolète n'est plus installé sur les nouveaux appareils. Les instances de ce composant déjà déployées ne sont pas impactées.

4.4 Conclusion

Dans ce chapitre, nous avons présenté un processus pour le déploiement automatique des systèmes répartis multi-échelles. Ce processus permet d'organiser les différentes étapes du déploiement. Dans un premier temps, le concepteur du déploiement spécifie les propriétés de déploiement. Puis, ces spécifications sont données en entrée d'un générateur de plan, avec l'état du domaine récupéré à début de l'exécution du déploiement. Ce plan de déploiement est ensuite réalisé : les composants sont installés et activés. Dans un second temps, le système de déploiement réagit autonomiquement au dynamisme du domaine de déploiement ainsi qu'à celui de l'application. Le déploiement incrémental permet de prendre en compte la dynamique de l'application, alors que le déploiement continu permet celle du domaine. Une boucle de contrôle autonomique permet de prendre en compte les variations de l'état du domaine (dégradation de l'état d'un appareil, disparition, etc.).

Contribution

Le processus de déploiement des systèmes répartis multi-échelles coordonne un certain nombre d'activités qui concernent la spécification du plan de déploiement, la génération et la réalisation du plan de déploiement initial, et la réalisation du déploiement dynamique. En particulier, le processus prend en compte l'ouverture du domaine de déploiement. La production du plan de déploiement est centralisée mais sa réalisation (déploiement initial et déploiement dynamique) est décentralisée et contrôlée dans le cadre d'une boucle « autonomique ». Pour cela, le processus s'appuie sur une infrastructure ouverte composée d'un support de spécification (un langage et son éditeur) et d'un middleware pour la mise en œuvre. En outre, le domaine et son état sont observés au moyen de sondes extraites de la spécification des propriétés spécifiées pour le plan de déploiement.

5 MuScADeL : Langage de spécification du plan de déploiement

Problématique

La conception du plan de déploiement est une activité particulière qui demande des moyens d'expression adéquats. Ces moyens sont donnés au concepteur sous la forme d'un langage dédié (DSL). Ce langage doit offrir un haut niveau d'abstraction qui permet de s'abstraire des problèmes de mise en œuvre. En outre, pour les systèmes multi-échelles, il faut pouvoir spécifier le plan sans connaître exactement le domaine ni désigner explicitement les différents appareils. En plus des composants, de leurs dépendances et de leurs contraintes d'exécution, le DSL doit permettre de spécifier les choix de conception ainsi que les sondes nécessaires à l'acquisition de l'état du domaine.

5.1 Introduction

Dans ce chapitre, nous raffinons les exigences concernant la conception du déploiement et présentons notre solution pour l'expression des propriétés de déploiement. Ces propriétés sont à la base de la production automatique d'un plan de déploiement adéquat et de son adaptation en cours d'exécution.

Comme nous l'avons présenté dans le chapitre précédent, dans le cadre des systèmes répartis multi-échelles, le concepteur de déploiement ne peut pas exprimer de plan de déploiement. La dynamique du domaine ne permet pas une connaissance précise des appareils impliqués dans le déploiement au moment de la conception. Le concepteur doit être capable de spécifier le déploiement de son application sans avoir une connaissance exacte du domaine de déploiement. Pour ce faire, un support d'expression offrant un haut niveau d'abstraction doit lui permettre de spécifier les différentes propriétés de son application (code, contraintes, versions, dépendances entre composants, etc.).

Le déploiement des systèmes répartis multi-échelles s'effectue en deux étapes, le

déploiement initial et le déploiement dynamique (cf. section 4.2). Un langage doit supporter l'expression de propriétés relatives à chacune des étapes : l'expression de propriétés à vérifier initialement, et l'expression des propriétés à vérifier à l'exécution du système. Afin de pouvoir générer le plan de déploiement initial, le support d'expression doit aussi permettre au concepteur d'exprimer des informations quant au domaine de déploiement : les informations utiles à la récupération des informations matérielles et logicielles spécifiques aux propriétés exprimées. De plus, le support d'expression doit aussi permettre la spécification d'informations relatives à la dynamique de l'application et la dynamique du domaine de déploiement. Le système déployé étant multi-échelle, le support d'expression doit permettre au concepteur d'exprimer des propriétés relatives aux aspects multi-échelles, spécifiques à son système.

La conception du déploiement requiert des compétences et des moyens particuliers. Ainsi, c'est un langage dédié (DSL, *Domain Specific Language*)à l'expression des propriétés de déploiement des systèmes répartis multi-échelles que nous proposons, le langage MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*). De manière générale, les DSLs présentent différents avantages. Ils utilisent les idiomes et les abstractions du domaine visé, et peuvent ainsi être utilisés par les experts du domaine. Ils sont légers, faciles à maintenir, portables et réutilisables. Ils sont le plus souvent bien documentés, cohérents et fiables, et optimisés pour le domaine visé [Van Deursen et al., 2000, Strembeck and Zdun, 2009, Tolvanen and Kelly, 2010].

Ce chapitre est organisé comme suit. Nous commençons par présenter dans la section 5.1.1 un état de l'art des DSLs pour le déploiement. Puis, nous exposons un exemple de système réparti multi-échelle dans la section 5.2. Dans la section 5.3, nous présentons les éléments de notre DSL MuScADeL. Enfin, nous présentons le framework de caractérisation multi-échelle dans la section 5.4 et montrons le lien entre ce framework et MuScADeL.

5

5.1.1 État de l'art des DSLs pour le déploiement

Les solutions de déploiement actuelles proposent différents formalismes pour l'expression des contraintes de déploiement, ainsi que pour les dépendances logicielles et matérielles des applications à déployer. Ces formalismes peuvent être des langages de description d'architecture (ADL, *Architecture Description Language*), des descripteurs de déploiement (par exemple en XML) ou des langages dédiés (DSL). Nous allons passer en revue quelques éléments de l'état de l'art des DSLs pour le déploiement.

Fractal Deployment Framework (FDF) [Flissi et al., 2008a] est un framework qui facilite le déploiement d'applications distribuées dans les systèmes connectés. FDF est composé d'un langage de description de déploiement, d'une librairie de composants de déploiement et d'un ensemble d'outils pour l'utilisateur. Le langage de description du déploiement de FDF permet au concepteur du déploiement de décrire la configuration du déploiement (la liste des logiciels à déployer et les appareils hôtes). FDF fournit aussi une interface graphique pour l'utilisateur qui lui permet de charger leurs configurations de déploiement, les exécuter et les gérer. L'unité de déploiement est une archive contenant le binaire de l'application et le descripteur du déploiement. La principale limitation de cet outil réside dans la saisie manuelle et statique des propriétés de déploiement. Bien qu'un plan de déploiement statique soit adapté à un environnement de déploiement tel que la grille, il n'est pas utilisable dans un environnement caractérisé par une topologie du réseau dynamique, ni dans un environnement ubiquitaire. Une autre limitation est qu'à l'exécution du déploiement, cet

outil n'offre pas de mécanisme de reconfiguration dynamique qui permettrait le traitement de pannes d'appareils ou de réseau.

Dearle *et al.* [Dearle et al., 2004, Dearle et al., 2010] présentent un framework pour la gestion autonomique du déploiement et la configuration des applications distribuées. Pour faciliter la tâche du concepteur du déploiement, ils ont défini un DSL, Deladas. Deladas permet de spécifier un ensemble de ressources disponibles ainsi qu'un ensemble de contraintes. Cette spécification permet de générer un plan de déploiement réalisable. L'approche à base de contraintes évite au concepteur de déploiement de définir spécifiquement l'emplacement de chaque composant, et ainsi réécrire l'ensemble du plan en cas de problème avec une ressource. Deladas ne permet pas d'exprimer des propriétés multi-échelles. L'ouverture du domaine de déploiement n'est pas prise en compte lors de l'expression, seul un ensemble de machines hôtes sont définies statiquement dans un fichier par le concepteur du déploiement. Le déploiement est par contre autonomique, à l'exécution, lorsque le middleware de déploiement détecte une violation de contraintes (dépendances entre composants), il essaie de la résoudre par une adaptation locale. Un nouveau plan de déploiement est alors calculé par le composant de gestion centralisée appelé MADME.

Matougui *et al.* [Matougui and Leriche, 2012] présentent un middleware qui réduit la tâche humaine lors de la mise en place du déploiement d'une application et qui permet de gérer les environnements sensibles aux pannes et aux changements. Ils utilisent un langage de haut-niveau à base de contraintes et un système d'agents autonomique pour l'établissement et le maintien du système déployé. Dans le DSL, appelé j-ASD, certaines expressions spécifiques à la gestion des situations autonomiques sont proposées. Le middleware est surtout spécifique aux environnements à large échelle et dynamique comme les grilles ou les systèmes pair à pair, dans la même échelle de réseau.

Sledviewsky *et al.* [Sledziewski et al., 2010] présentent une approche qui incorpore les langages dédiés pour le déploiement logiciel et plus spécifiquement le déploiement dans le Cloud. Premièrement, le développeur définit un DSL pour décrire un modèle de l'application en l'utilisant. Puis, l'application est décrite en utilisant ce DSL pour finalement être traduit automatiquement dans du code spécifique et déployé sur le Cloud. Cette approche est spécifique au déploiement d'un application web sur le Cloud. Cela met en avant le besoin de faciliter la tâche du concepteur de déploiement, et que l'utilisation d'un DSL est une solution pour ce problème.

5.2 Déploiement d'un système multi-échelle : l'exemple de 4ME

Pour illustrer notre présentation de MuScADeL, nous prenons l'exemple du déploiement d'une partie de l'application de 4ME. Nous nous intéresserons au déploiement de sept types composants :

— Bike pour la gestion des stations de vélo,
— BikeAvail pour l'agrégation de la disponibilité des vélos,
— Stat pour les statistiques,
— GUI pour l'interface graphique destinée aux smartphones,
— IM pour la gestion de la messagerie instantanée sur les smartphones,
— IMHist pour la gestion de l'historique de la messagerie instantanée,
— RoutePlanner (ou RP) pour le calcul des trajets.

Chaque composant a des **contraintes d'exécution propres** (par ex. mémoire minimale, système d'exploitation, etc.). Le concepteur du déploiement doit pouvoir exprimer ces contraintes ainsi que d'autres propriétés relatives à la distribution des composants, c'est-à-dire ses **exigences en matière de déploiement**. Les **propriétés de déploiement** sont cet ensemble de contraintes et d'exigences définies sur les composants du système. Par exemple, le concepteur du déploiement peut vouloir spécifier les exigences suivantes :

— un composant de type Bike doit être déployé sur tous les appareils de station de vélo ;
— un composant de type BikeAvail doit être déployé sur un appareil connecté *via* le réseau SigFox [1] pour 3 composants de type Bike déployés ;
— un composant de type Stat doit être déployé sur 10 à 30 Cloudlet ;
— un composant de type GUI doit être déployé sur tous les smartphones appartenant au domaine, y compris sur tout nouveau smartphone entrant dans le domaine ;
— un composant de type IM doit être déployé sur tous les smartphones de chaque groupe de personnes ;
— un composant de type IMHist doit être déployé sur un smartphone pour chaque groupe de personnes ;
— deux composants de type RoutePlanner doit être déployé sur un serveur sur le Cloud géré par l'entreprise de station de vélo.

De plus, certains composants peuvent imposer certaines contraintes relatives à leur fonctionnement :

— un composant GUI a besoin d'un écran de taille minimale de 4 pouces ;
— un composant IM a besoin que le composant GUI soit installé et activé localement et de 80Mo de libre à l'installation ;
— un composant Stat a besoin de disposer de 1 Mo de mémoire vive libre et d'un processeur d'au moins 1Ghz.

La figure 5.1 illustre cet exemple.

5.3 Le DSL MuScADeL

Dans cette section, nous décrivons le DSL MuScADeL en nous appuyant sur l'exemple de déploiement du système réparti multi-échelle présenté précédemment. La grammaire, définie utilisant la syntaxe EBNF, est disponible dans l'annexe B. Le code est présenté en cinq extraits présentant respectivement les composants (listing 5.1), les critères (listing 5.2), les sondes (listing 5.3), les sondes multi-échelles (listing 5.4) et les exigences de déploiement (listing 5.5). L'exemple complet du descripteur MuScADeL est disponible à l'annexe A.

5.3.1 Composant

L'unité de déploiement est le composant. Le descripteur (ou code) MuScADeL liste les types de composants du système, utilisant le mot-clé `Component` (listing 5.1). Une description d'un type de composant comporte plusieurs champs. Le champ `Version` est utile pour l'activité de mise à jour. Le champ `URL` spécifie l'adresse à laquelle le code du composant est téléchargeable. Le champ `DeploymentInterface` spécifie l'interface de contrôle du composant. Il est essentiel pour permettre l'interaction entre le composant et le système de

1. SigFox propose un réseau cellulaire pour la connexion de petits objets connectés [Sig].

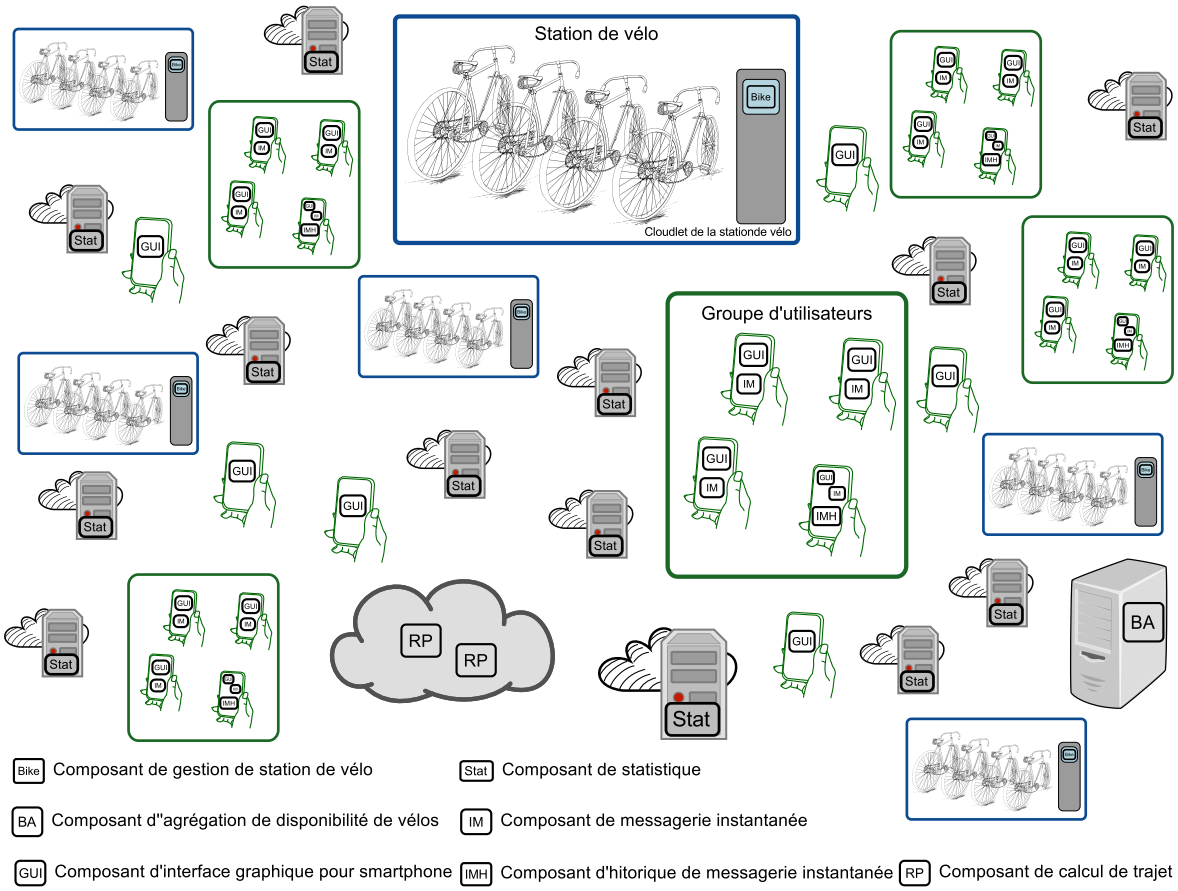


Figure 5.1 – Exemple d'un déploiement d'un système multi-échelle

déploiement : ce dernier doit interagir avec le composant pour le configurer, l'activer, l'administrer en cours d'exécution et le désactiver. Le champ `Dependency` liste les composants requis : à l'installation du composant, si un composant qu'il requiert n'est pas déployé, le système de déploiement doit le déployer sur l'appareil. Des contraintes peuvent être ajoutées à un composant. Le champ `Constraint` liste les conditions logicielles et matérielles que le déploiement doit respecter, définies avec le mot-clé `BCriterion`, par exemple à la ligne 2 du listing 5.2. Par défaut, les contraintes doivent être satisfaites à la fois lors de la génération du plan de déploiement et lors de l'exécution du système (le système de déploiement doit vérifier dynamiquement qu'il n'y a pas de violation de contraintes). Des contraintes qui doivent être satisfaites uniquement à l'installation peuvent être définies utilisant le mot-clé `InitOnly`, elles ne seront pas vérifiées à l'exécution du système (ligne 16 du listing 5.1).

```
1  Component Bike {
2      Version 1
3      URL "http://income.fr/4ME/bike.jar"
4  }

6  Component BikeAvail {
7      Version 1
8      URL "http://income.fr/4ME/bikeAvail/jar"
9      DeploymentInterface fr.enac.plop.DIimpl
10 }
```

```
12 Component GUI {
13     Version 1
14     URL "http://income.fr/4ME/gui.jar"
15     Constraint RAM80
16     InitOnly Constraint Screen
17 }

19 Component IM {
20     Version 1
21     URL "http://income.fr/4ME/im.jar"
22     Dependency GUI
23     InitOnly Constraint RAM80
24 }

26 Component Stat {
27     Version 2
28     URL "http://income.fr/4ME/stat.jar"
29     Constraint CPURAM
30 }
```

Listing 5.1 – `Component` – Spécification des composants dans MuScADeL

Par exemple, le type de composant `GUI` est en version 1, disponible à l'adresse `http://income.fr/4ME/gui.jar`. À son installation, l'écran de l'appareil doit être d'une taille d'au moins 4 pouces et lors de l'installation et l'exécution du composant, l'appareil doit disposer d'au moins 80Mo de mémoire vive disponible.

5.3.2 Critère

De manière générale, les critères sont des conditions à respecter. Ils sont utilisés pour la spécification de contraintes propres à un composant ou la spécification d'exigences de déploiement. Le mot-clé `BCriterion` définit un critère (ici, il s'agit de critères de base par opposition aux critères multi-échelles, introduits dans la section 5.3.5). Un critère est une condition, une conjonction ou une disjonction de conditions portant sur des valeurs sondées, comme `CPURAM` (listing 5.2, ligne 5). Il existe deux types de conditions, l'une concernant l'existence ou l'activité d'une sonde, l'autre concernant la valeur donnée d'une sonde.

Dans le premier cas, la condition est composée du nom de la sonde et des mots-clés `Exists` ou `Active`, dont les méthodes sont définies dans toutes les interfaces des sondes. Par exemple, dans le listing 5.2, à la ligne 2, la sonde utilisée est `SigFox`, et la condition utilise les méthodes définies par défaut `Exists` et `Active`.

Si la condition est une condition valuée, elle est composée du nom de la sonde, de la méthode à appeler, d'un comparateur et de la valeur. Dans ce cas, la méthode est spécifique à la sonde et est définie dans l'interface de la sonde. Par exemple, dans le listing 5.2, à la ligne 7, la sonde utilisée est `RAM`, la méthode utilisée est `free` et la valeur comparée est le nombre `1`, pour 1Mo. Un critère peut être utilisé pour définir une contrainte portant sur un composant (listing 5.1, ligne 15) ou une exigence de déploiement (listing 5.5, ligne 7). Le critère `LinuxOrAndroid` (ligne 10) utilise l'opérateur `|` de disjonction entre deux conditions. Ce critère défini que l'appareil doit avoir comme système d'exploitation Linux ou Android. La disjonction des conditions est prioritaire à la conjonction des conditions.

```
1 BCriterion SigFoxActive {
2   SigFox Exists; SigFox Active;
3 }
```

```
5  BCriterion CPURAM {
6    CPU.proc > 1;  // at least 1Ghz of CPU
7    RAM.free > 1;  // at least 1Mb free RAM
8  }

10 BCriterion LinuxOrAndroid {
11     OS.name = "Linux"
12     | OS.name = "Android";
13 }
```

Listing 5.2 – `BCriterion` – Spécification des critères dans MuScADeL

5.3.3 Sonde

Les sondes logicielles sont utilisées dans la définition des conditions des critères. Elles permettent de récupérer des informations précises concernant l'appareil hôte. Les sondes logicielles sont définies en utilisant mot-clé `Probe`. Elle comporte deux champs obligatoires. Le premier champ est `ProbeInterface`, qui spécifie l'interface de la sonde. Le second est le champ `URL`, qui indique l'adresse de téléchargement du code de la sonde.

```
1  Probe SigFox {
2      ProbeInterface fr.irit.sigfox.DIImpl
3      URL "http://irit.fr/INCOME/SigFoxProbe.jar"
4  }
```

Listing 5.3 – `Probe` – Spécification des sondes dans MuScADeL

5.3.4 Sonde multi-échelle

La conception du système réparti multi-échelle est basée sur une caractérisation multi-échelle du système. Cette caractérisation définit les points de vue, les dimensions et les échelles de l'application. Les sondes multi-échelles permettent de récupérer des informations multi-échelles sur l'appareil hôte. Les sondes multi-échelles sont définies en utilisant le mot-clé `MultiScaleProbe`. Comme `Probe`, elle n'a que deux champs, `MultiScaleProbeInterface` et `URL`, qui spécifient respectivement l'interface de la sonde multi-échelle et l'adresse à laquelle son code est téléchargeable.

Une sonde est définie par point de vue. Les sondes permettent d'identifier la dimension, l'échelle et l'instance d'échelle d'un appareil donné. L'instance d'une échelle est la valeur spécifique à l'appareil de l'échelle à laquelle il appartient. Un mot-clé spécifique est nécessaire aux sondes multi-échelles car elles ont un traitement opérationnel spécifique lors de la récupération de l'état du domaine (cf. sections 8.2.1.1 et 8.2.5). Le code MuScADeL des sondes multi-échelles peut être généré par le framework MuSCa (cf. section 5.4.1).

```
1  MultiScaleProbe Device {
2    MultiScaleProbeInterface eu.telecom-sudparis.DeviceProbeImpl
3    URL "http://it-sudparis.eu/INCOME/DeviceProbe.jar"
4  }
5  MultiScaleProbe Administration {
6    MultiScaleProbeInterface eu.telecom-sudparis.AdminProbeImpl
7    URL "http://it-sudparis.eu/INCOME/AdminProbe.jar"
8  }
```

Listing 5.4 – `MultiScaleProbe` – Spécification des sondes multi-échelles dans MuScADeL

5.3.5 Déploiement multi-échelle

Une fois tous ces éléments définis, les exigences de déploiement du système multi-échelle peuvent être spécifiées. L'opérateur `@` permet de spécifier une conjonction d'exigences spécifiques à un composant. L'ensemble des exigences peuvent prendre plusieurs formes, tel que présenté dans le listing 5.5.

Le mot-clé `AllHosts` permet de circonscrire le domaine de déploiement : la ligne 2 permet de définir que le déploiement doit s'effectuer sur tous les appareils satisfaisant le critère `LinuxOrAndroid`.

```
1  Deployment {
2      AllHosts LinuxOrAndroid;

4      Bike @ All, Administration.Level.Service("Toulouse.SharingBikes"),
5              Device.Type.Cloudlet;
6      BikeAvail @ 1/5 Bike, SigFoxActive;
7      Stat @ 10..30, Device.Type.Cloudlet;
8      GUI @ All, Device.Types.Smartphone;
9      IM @ All, Device.Type.Smartphone,
10                 User.NumberOfUsers.Group;
11     IMHist @ Device.Type.Smartphone,
12                      Each User.NumberOfUsers.Group;
13     RoutePlanner @ 2, SameValue Administration.level.service(Bike),
14                     Device.StorageCapacity.Giga;
15 }
```

Listing 5.5 – `Deployment` – Spécification des exigences de déploiement dans MuScADeL

Le concepteur doit néanmoins pouvoir identifier de désigner des parties du domaine, sans qu'il ne puisse désigner explicitement la totalité des appareils. Ce sont les notions d'échelle et d'instance d'échelle qui apportent une solution à ce problème et permettent de spécifier le déploiement d'un composant dans telle ville, sur un tel réseau local, sur des appareils appartenant à tel domaine administratif, etc. Le listing 5.5 montre un certain nombre d'exigences de déploiement à une certaine échelle :

- Le composant `Bike` doit être déployé sur tous les appareils qui font partie de *(i)* l'échelle `Cloudlet` dans la dimension `Type` du point de vue `Device`, et *(ii)* sont administrés par le service `"Toulouse.SharingBikes"` — instance `"Toulouse.SharingBikes"` dans l'échelle `Administration.Level.Service` — (lignes 4 et 5) ;
- Les composants `BikeAvail` doivent être déployés sur des appareils satisfaisant le critère `SigFoxActive`, l'expression du ratio `1/5` permettant de spécifier qu'un composant `BikeAvail` doit être déployé pour chaque lot de cinq composants `Bike` déployés (ligne 6) ;
- Entre dix et trente composants `Stat` doivent être déployés sur des appareils appartenant à l'échelle `Device.Type.Cloudlet` (ligne 7) ;
- Le composant `GUI` doit être déployé sur tous les appareils qui appartiennent à l'échelle `Device.Type.Smartphone`, c'est-à-dire sur les smartphones du domaine de déploiement, y compris ceux entrant dans le domaine de déploiement à l'exécution du système, utilisant le mot-clé `All` (ligne 8) ;
- Le composant `IMHist` doit être déployé sur un smartphone dans chaque instance de l'échelle `User.NumberOfUsers.Group` (mot-clé `Each`), c'est-à-dire, un smartphone de chaque groupe de personnes (lignes 11 et 12) ;
- Le composant `RoutePlanner` doit être déployé sur deux appareils *(i)* ayant une

grande puissance de calcul (appartenant à l'échelle `Device.StorageCapacity.Giga`), et *(ii)* étant administré par le même service d'entreprise que le composant `Bike`, en utilisant le mot-clé `SameValue` (lignes 13 et 14).

Le descripteur MuScADeL doit contenir au moins une définition d'un composant et l'expression d'un déploiement. Les autres sections sont optionnelles. Le code peut être découpé en plusieurs fichiers, le mot-clé `Include` permet d'inclure d'autres fichiers.

5.3.6 MuScADeL et la gestion de la dynamique

Certaines constructions du DSL sont adaptées à l'expression de propriétés relatives à la dynamique et à l'ouverture. Par défaut, les contraintes doivent être satisfaites initialement, dès la génération du plan de déploiement, puis durant l'exécution de l'application. Elles doivent donc être vérifiées dynamiquement. Le mot-clé `InitOnly` permet de spécifier qu'une (ou plusieurs) contraintes doivent être satisfaites initialement mais non nécessairement à l'exécution.

D'autre part, lorsque le déploiement est spécifié, dans la section `Deployment`, le mot-clé `All` permet d'exprimer qu'un composant doit être déployé sur un sous-domaine qui satisfait dynamiquement la propriété exprimée. Dans l'exemple, le composant `GUI` doit être déployé sur tous les smartphones du domaine, incluant ceux qui entrent dans le domaine alors que l'application s'exécute. Il en est de même pour la propriété de ratio. Ainsi, le plan de déploiement évolue dynamiquement en fonction des entrées et des sorties des appareils composant le domaine de déploiement.

5.4 Caractérisation multi-échelle et MuScADeL

L'expression du déploiement s'appuie sur les caractéristiques multi-échelles du système déployé. En dehors de la conception du déploiement, une analyse du système permet d'identifier ces caractéristiques. Cette caractérisation multi-échelle s'appuie sur un framework, appelé MuSCa (***Mu**lti**S**cale **Ca**racterization framework*), défini par S. Rottenberg dans le cadre de sa thèse [Rottenberg, 2015] (voir également [Rottenberg et al., 2014]).

5.4.1 Caractérisation multi-échelle

Afin de formaliser le processus de caractérisation multi-échelle, une approche basée sur l'ingénierie dirigée par les modèles (IDM) a été suivie. La figure 5.2 montre les liens entre les niveaux de modélisations de l'IDM et les niveaux MuSCa. Un métamodèle MuSCa (niveau M2) a été défini en utilisant le méta-métamodèle Ecore [Budinsky et al., 2008] (niveau M3). Les classes du métamodèle MuSCa représentent les concepts multi-échelles. Avec MuSCa, les modèles de caractérisation (niveau M1) peuvent être définis. Cette caractérisation peut être utilisée pour un ou plusieurs systèmes réels (niveau M0). L'approche dirigée par les modèles est aussi utilisée afin de produire automatiquement des éléments logiciels, par exemple des sondes logicielles utilisées pour la détection d'échelles.

Le métamodèle MuSCa est représenté dans la figure 5.3. Ce métamodèle est basé sur le vocabulaire utilisé pour la caractérisation multi-échelle, c'est-à-dire point de vue, dimension, mesure, ensemble d'échelles et échelle. Une instance de MSCharacterization est

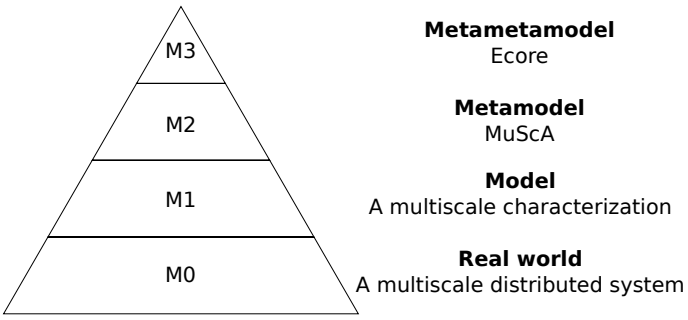


Figure 5.2 – MuSCa : niveaux de modélisation de l'ingénierie dirigée par les modèles [Rottenberg, 2015]

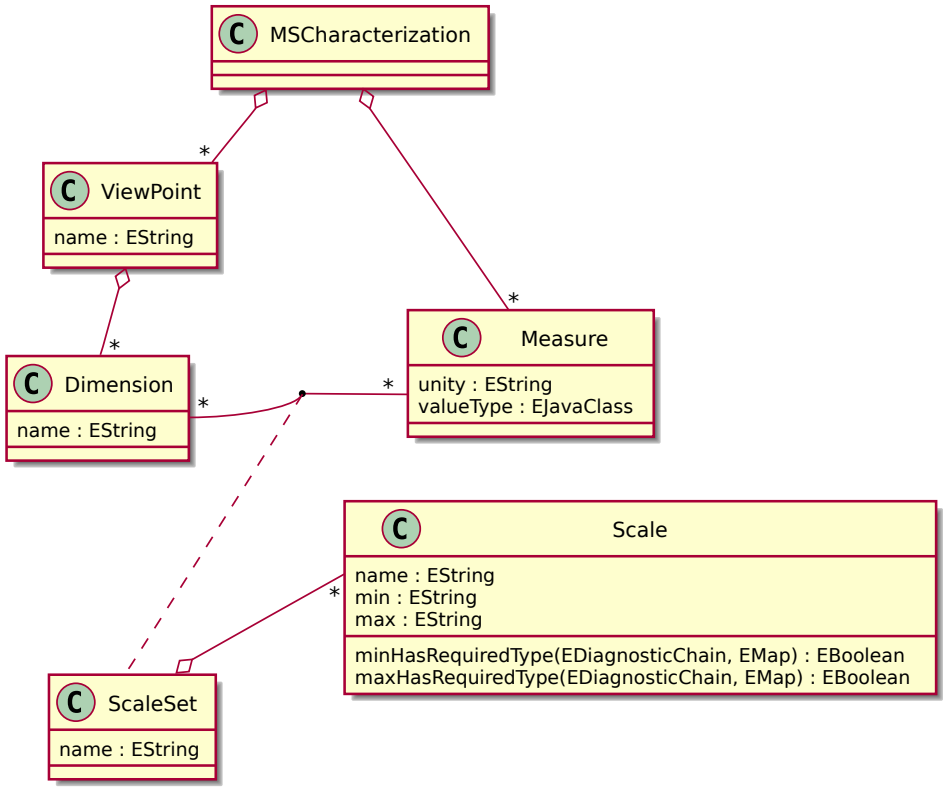


Figure 5.3 – MuSCa : métamodèle de caractérisation multi-échelle [Rottenberg, 2015]

le résultat d'un processus de caractérisation. Une caractérisation contient plusieurs Viewpoint. Chaque point de vue détermine une vue restreinte du système étudié. Dans un point de vue donné, la vue du système est étudiée à travers différentes Dimension, qui sont des caractéristiques mesurables des éléments de la vue. Par exemple, pour le point de vue Device (appareil), les appareils du système peuvent être analysés à travers la dimension StorageCapacity (capacité de stockage, niveau M1). Une Dimension étant mesurable, elle peut être associée à différentes Measure (mesure). Par exemple, au niveau M1, la dimension StorageCapacity peut être mesuré suivant la mesure Bytes (octets) ou KiloBytes (kilo-octets). Pour l'association d'une dimension à une mesure, le concepteur définit un ScaleSet (ensemble d'échelles), qui est un ensemble ordonné d'échelles significatives du système étudié. Pour des mesures numériques, une Scale est définie par ses limites minimale, min, et maximale, max. Pour certains points de vue, le système peut avoir différentes instances dans une

échelle. Par exemple, pour le point de vue Geography, dans la dimension `Administration`, l'échelle Town (ville, niveau M1) peut avoir plusieurs instances (niveau M0), c'est-à-dire les différentes villes dans lesquelles des entités du système sont présentes.

5.4.2 MuSCa et MuScADeL

Nous avons conçu le DSL de manière à répondre aux problématiques multi-échelles. Des mots-clés spécifiques sont intégrés au langage, comme `MultiScaleProbe`, pour l'expression des sondes multi-échelles. MuScADeL permet aussi l'expression d'exigences relatives aux échelles. Étant donné qu'un code MuScADeL est lié à un modèle spécifique MuSCa, l'éditeur de MuScADeL vérifie que les dimensions et les échelles sont bien conformes à celle définies dans ce modèle. De plus, les exigences relatives aux échelles sont vérifiées à l'exécution par les sondes multi-échelles générées par le framework MuSCa.

La figure 5.4 est un diagramme de classe UML qui montre les métamodèles MuSCa et MuScADeL, et leur liens. Pour des raisons de lisibilité, seulement une partie des métamodèles est montrée. Le métamodèle MuScADeL est limité à la partie critère des exigences de déploiement.

Dans le métamodèle MuScADeL, Criterion est spécialisé en MSCriterion pour l'expression de critères multi-échelles. Cet élément est spécialisé en SameValue, Each et Simple pour l'expression des différentes exigences de déploiement relatives aux échelles exprimées dans la section 5.2. Contrairement aux critères basiques (`BCriterion`) il n'existe pas de mot-clé pour définir les critères multi-échelles, MSCriterion dans le métamodèle. Ces critères sont exprimés tel quel dans l'expression des exigences de déploiement, tel que vu dans le listing 5.5. Par exemple, Each exprime la notion d'un composant sur un appareil par instance d'échelle. Un critère multi-échelle peut être exprimé selon une échelle (niveau M1) ou une instance d'échelle (niveau M0). Par exemple, il est possible d'exprimer qu'un composant doit être déployé sur appareil appartenant à un service d'entreprise (niveau M1, `Administration.Level.Service`) ou à un service d'entreprise particulier (niveau M0, `Amdinistration.Level.Service("Toulouse.SharingBikes")`).

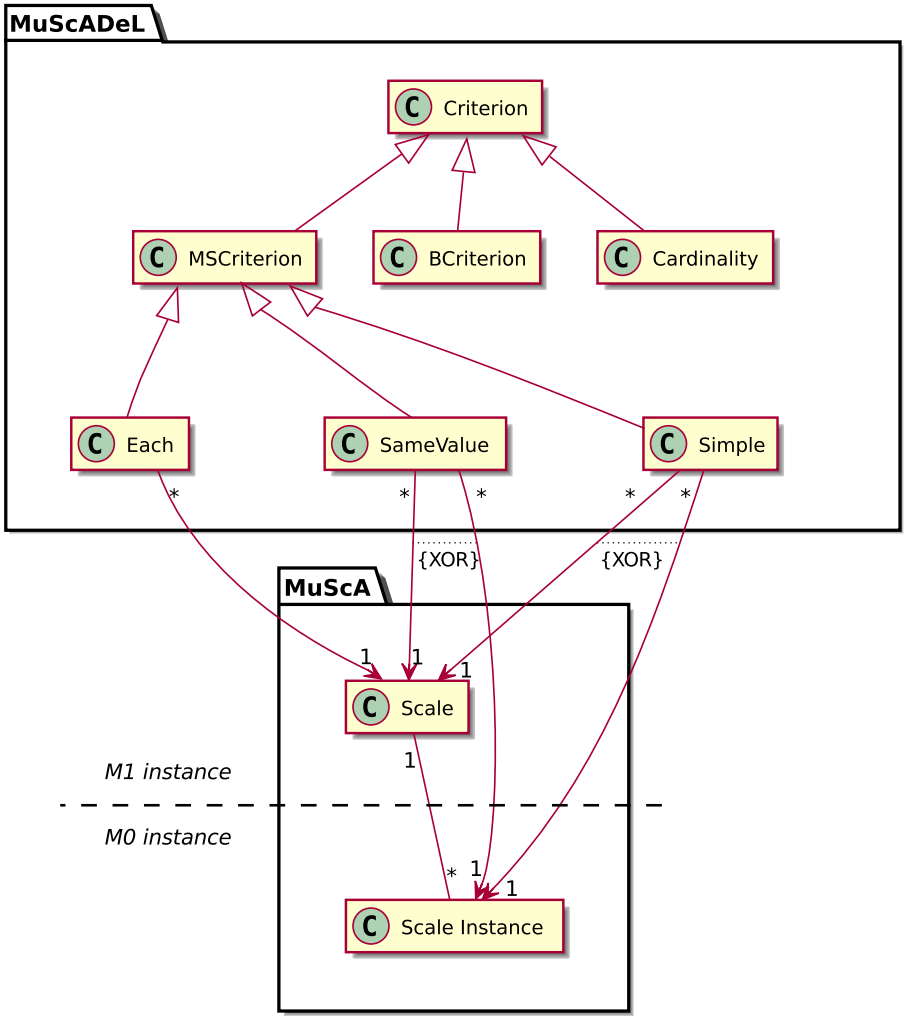


Figure 5.4 – Partie du métamodèle MuScADeL lié au métamodèle MuSCa

5.5 Conclusion

Dans ce chapitre, nous avons présenté MuScADeL, une proposition de langage dédié au déploiement de systèmes répartis multi-échelles. L'utilisation d'un DSL permet de fournir un moyen d'expression du déploiement avec haut niveau d'abstraction. MuScADeL permet au concepteur non expert en déploiement (mais expert dans le domaine de l'application à déployer) de spécifier les composants de l'application et leurs relations, ainsi que les propriétés de déploiement. Ces propriétés sont des contraintes d'exécution propres aux composants et des exigences de déploiement (qui sont les choix du concepteur). Certaines expressions du langage sont spécifiques à la dynamique du domaine de déploiement. Ainsi, le concepteur peut directement donner des directives pour la prise en compte du déploiement continu de l'application à déployer. De plus, MuScADeL permet la spécification d'exigences en fonction des caractéristiques multi-échelles du système à déployer, ainsi que la définition de sondes et de critères multi-échelles. De par le couplage entre MuSCa et MuScADeL, ces exigences sont vérifiées et validées par l'éditeur, et afin de simplifier la tâche au concepteur, le code MuScADeL relatif aux sondes multi-échelles est généré par MuSCa.

Contribution

Le langage dédié MuScADeL permet spécifier le déploiement de systèmes repartis multi-échelles. Le concepteur du déploiement peut exprimer les contraintes d'exécution des composants ainsi que ses exigences en matière de distribution, de nombre et de dynamique du domaine. Le concepteur peut aussi exprimer les éléments qui vont permettre l'acquisition de l'état du domaine. Pour pouvoir spécifier le déploiement sans connaissance exacte du domaine et désigner des parties du domaine, on s'appuie sur les échelles identifiées dans le système. MuScADeL est lié au métamodèle MuSCa, ce qui permet une expression valide des propriétés relatives aux échelles.

6 Déploiement automatique : architecture et formalisation

Problématique

Le domaine de déploiement étant de grande taille, dynamique et incomplètement connu à la conception, le déploiement initial doit être automatisé : le système de déploiement doit acquérir l'état du domaine et générer un plan de déploiement conforme aux spécifications exprimées, puis le réaliser. Pour cela, les spécifications doivent être traitées automatiquement et une architecture adéquate du système de déploiement doit être conçue.

6.1 Introduction

Dans notre processus, nous proposons que la récupération de l'état du domaine, ainsi que la génération du plan de déploiement puis sa réalisation soient supportées de manière automatique, c'est-à-dire avec une intervention humaine minimale. Dans ce chapitre, nous présentons les éléments de notre solution pour l'automatisation du déploiement. Ces éléments supportent la mise en œuvre d'une partie du processus de déploiement présenté dans le chapitre 4, plus précisément dans les sections 4.3.1 à 4.3.3.

Ce chapitre présente de manière abstraite l'architecture. La présentation concrète de l'architecture, avec les choix technologiques et leurs impacts sont présentés dans le chapitre 8.

Nous commençons par décrire l'architecture du bootstrap et du système de déploiement au moment de la production du plan (cf. section 6.2). Puis, nous présentons la formalisation des propriétés de déploiement sous la forme de contraintes, en vue de les faire traiter par un solveur de contraintes (cf. section 6.3), puis nous illustrons la génération du plan de déploiement en nous basant sur l'exemple donné dans la section 5.2. Enfin, nous présentons l'architecture du système de déploiement au moment de l'installation et de l'activation de l'application déployée (cf. section 6.4).

6.2 Architecture initiale du système de déploiement

6.2.1 Bootstrap

Le bootstrap est un programme d'amorce installé sur chaque appareil, sur lequel s'appuie le système de déploiement.

Pour la conception du bootstrap, nous avons choisi une architecture composant-conteneur. Il est donc un conteneur de composants initialement composé de trois composants : Main, DomainManagement, et BasicProbes. Le composant Main est le point d'entrée du bootstrap. Le composant DomainManagement permet la liaison avec le gestionnaire de domaine de déploiement. Le composant BasicProbes contient un ensemble de sondes basiques (cf. section 6.2.2.1) présentes systématiquement sur chaque appareil. Le contenu du bootstrap (l'ensemble des composants) évolue au fur et à mesure de l'avancement du processus de déploiement, par exemple pour s'enrichir de nouvelles sondes spécifiques à un appareil où à la supervision de contraintes multi-échelles.

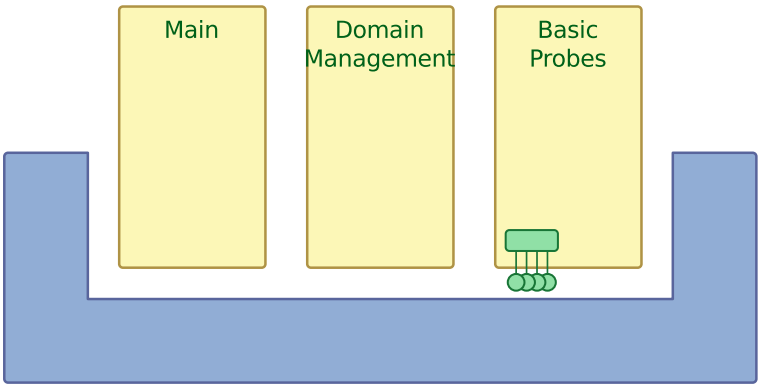


Figure 6.1 – Architecture du bootstrap

6.2.2 Production du plan de déploiement

6.2.2.1 Sondes

Les sondes permettent de récupérer des informations propres à chaque appareil. Elles sont nécessaires à plusieurs stades : lors du déploiement initial et de la production du plan, et à l'exécution pour les activités de déploiement dynamique. Il existe différents types de sondes :

— les sondes « basiques » (ou génériques) présentes dans le bootstrap,
— les sondes « spécifiques », définies par le concepteur dans le descripteur MuScADeL, nécessaires pour constituer un état du domaine spécifique au déploiement en cours de réalisation.

Les sondes basiques et les sondes spécifiques sont organisées en deux composants distincts (cf. figure 6.2). Le composant des sondes spécifiques est transféré sur les appareils du domaine puis installé et activé. C'est ce composant qui est désigné « composant des sondes spécifiques (version 1) » (la justification à propos de ces versions est donnée en section 6.4.2).

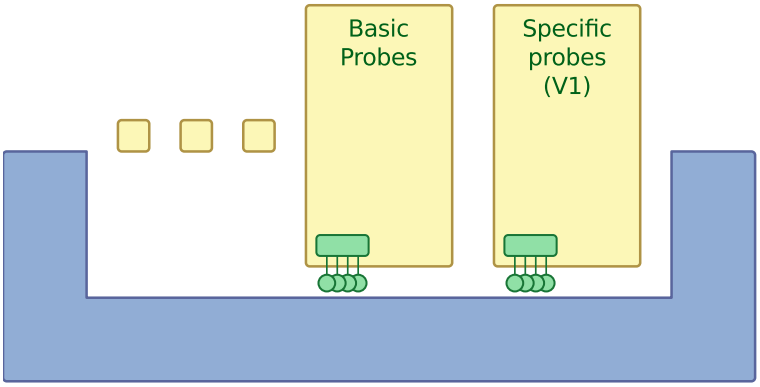


Figure 6.2 – Détail des composants de sondes

6.2.2.2 Construction de l'état du domaine

La figure 6.3 présente la phase de production du plan de déploiement d'un point de vue séquentiel. Les étapes sont les suivantes :

① Les appareils faisant partie du domaine de déploiement s'enregistrent auprès du gestionnaire du domaine.
② Le moniteur du déploiement récupère la liste des appareils enregistrés.
③ Le moniteur de déploiement envoie à ces appareils un composant contenant l'ensemble des sondes spécifiques (version 1).
④ L'appareil installe et active le composant des sondes spécifique (version 1), et lance le sondage local.
⑤ Les appareils renvoient les valeurs au moniteur de déploiement (cf. section 6.3.3). L'ensemble des données obtenues, qui constitue l'état du domaine, est traité (transformation, agrégation) par le moniteur du déploiement puis donné en entrée de l'activité de génération du plan de déploiement (cf. figure 4.6).

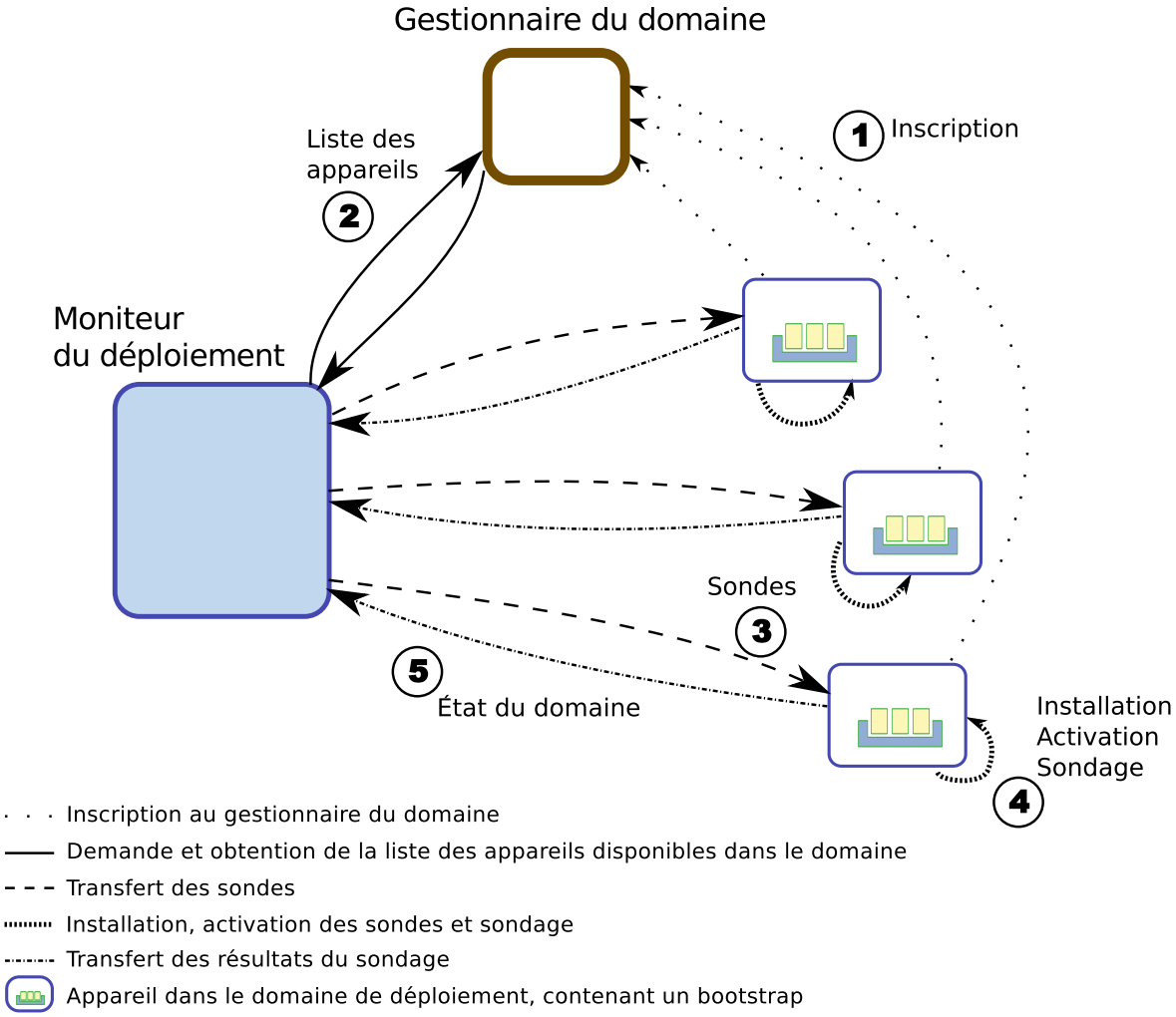


Figure 6.3 – Étapes de la récupération de l'état du domaine

6.3 Formalisation des contraintes et génération du plan

L'étape qui suit la récupération de l'état du domaine est celle de génération du plan de déploiement. Comme les spécifications exprimées dans le descripteur MuScADeL constituent un ensemble de contraintes que le plan de déploiement doit respecter étant donné l'état du domaine, il est naturel d'utiliser un solveur de contraintes pour produire le plan de déploiement (le choix du solveur est discuté en section 8.2.6.1). Nous montrons ici comment les propriétés peuvent être traduites en contraintes.

6.3.1 Données et structures de données

6.3.1.1 Données en entrée

L'analyse du descripteur MuScADeL a permis d'identifier un certain nombre de propriétés que le système doit posséder. Elle a produit une matrice $Comp$ de type $Composant * Propriété$ telle que

— $Comp(i, j) = 1$ si le déploiement du composant $Composant_i$ est contraint par la pro-

priété $Propriété_j$,
— $Comp(i,j) = 0$ sinon.

Les composants dont nous avons spécifié qu'ils sont requis par des composants à déployer sont pris en compte ici et intégrés à la matrice $Comp$. Ici, nous ne considérons que des propriétés simples, c'est-à-dire ne concernant qu'un seul composant. D'autres propriétés, qui ne sont pas prises en compte dans la matrice $Comp$, restent à considérer.

D'autre part, indépendamment de l'analyse du descripteur MuScADeL, l'analyse de l'état du domaine a produit une matrice Dom de type $Appareil * Propriété$ telle que
— $Dom(i,j) = 1$ si l'appareil $Appareil_i$ satisfait la propriété $Propriété_j$,
— $Dom(i,j) = 0$ sinon.

Les sondes, y compris les sondes multi-échelles, sont utilisées pour construire la matrice Dom. De manière générale, les mesures prises par les sondes sur les appareils font aussi partie de l'état du domaine. Elles sont fournies sous la forme d'une table associant appareil et mesure.

6.3.1.2 Données en sortie

Le plan de déploiement produit par le solveur se présente sous la forme d'une matrice d'obligation $Oblig$ de type $Composant * Appareil$, qui définit le placement des composants, telle que
— $Oblig(i,j) = 1$ si le composant $Composant_i$ doit être déployé sur l'appareil $Appareil_j$,
— $Oblig(i,j) = 0$ sinon.

6.3.2 Les propriétés de déploiement

6.3.2.1 Contraintes et exigences

Une matrice des possibilités $SatVar$, de type $Composant * Appareil$, est construite. Chaque coefficient de $SatVar$ est une variable qui peut prendre sa valeur dans $\{0,1\}$.

À partir des matrices $Comp$ et Dom, des contraintes sont ajoutés sur certains coefficients. Ces contraintes sont l'affectation à 0 du coefficient, correspondant à une impossibilité pour un appareil d'héberger le composant. Cela se traduit par la contrainte suivante[1] (nb_app et nb_comp correspondent respectivement au nombre d'appareils et au nombre de composants impliqués dans le déploiement) :

$$\forall i \in \{1,..,nb\_comp\}, \forall j \in \{1,..,nb\_app\}$$
$$Comp(i) \cdot Dom(j) = \vec{0} \implies SatVar(i,j) = 0 \quad (6.1)$$

Ici, $Comp(i)$ et $Dom(j)$ représentent respectivement les lignes i et j des matrices $Comp$ et Dom, l'opérateur $\cdot$ construit une ligne composée des produits deux à deux des éléments de deux lignes données, et $\vec{0}$ représente le vecteur nul.

1. Par convention, les indices de lignes et de colonnes des matrices et des tableaux commencent à 1.

6.3.2.2 Nombre d'instance d'un composant

Cardinalité Pour chaque composant (une ligne de la matrice *SatVar*), le nombre d'instances est défini par la valeur de la somme des éléments de la ligne. Nous construisons ainsi une contrainte pour chaque composant en fonction du nombre d'instances.

Ainsi, si le composant C_k doit être déployé sur n_k appareils, cela se traduirait par la contrainte :

$$\sum_{j=1}^{nb\_app} SatVar(k,j) = n_k \tag{6.2}$$

Si le composant C_k doit être déployé sur n_k à m_k appareils, cela se traduirait par la contrainte :

$$n_k \leq \sum_{j=1}^{nb\_app} SatVar(k,j) \leq m_k \tag{6.3}$$

All La cardinalité All spécifie qu'un composant doit être déployé sur tous les appareils qui peuvent l'héberger. Il faut maximiser le nombre d'appareils qui peuvent héberger le composant. L'expression `Ck @ All` se traduirait par la formule suivante :

$$\max_{\substack{j\in\{1,..,nb\_app\} \\ SatVar(k,j)}} \left(\sum_{i=1}^{nb\_app} SatVar(k,i) \right) \tag{6.4}$$

Ratio Un ratio entre les nombres d'instances de différents composants peut se traduire en s'appuyant sur les mêmes principes, mais en associant plusieurs lignes de *SatVar*. L'expression `Ck @ n/m Cl` se traduirait par la contrainte :

$$\sum_{i=1}^{nb\_comp} SatVar(k,i) = n \times \left\lfloor \frac{\sum_{i=1}^{nb\_comp} SatVar(l,i)}{m} \right\rfloor \tag{6.5}$$

où $\lfloor \cdot \rfloor$ désigne la partie entière.

Dépendance Lors de la description d'un composant, le concepteur du déploiement peut préciser si ce composant est dépendant d'un autre. Dans ce cas, il faut que les deux composants soient sur le même appareil. Soit C_k dépendant de C_l, cela se traduirait par la formule suivante :

$$\forall i \in \{1,..,nb\_comp\} SatVar(k,i) = 1 \implies SatVar(l,i) = 1 \tag{6.6}$$

6.3.2.3 Propriétés multi-échelles

Composants dépendants Les propriétés à caractère multi-échelle exprimées au moyen des mots-clés `SameValue` et `DifferentValue` portent sur plusieurs composants à la fois. Ces propriétés expriment des conditions nécessaires pour déploiement des composants, qui sont contrôlées à partir des valeurs fournies par la sonde multi-échelle concernée. Par exemple, l'expression `Ck @ SameValue Some.MS.Scale(Cl)` exprime que les composant C_k et C_l doivent être dans la même instance d'échelle de `Some.MS.Scale`. Soit *MSProbe* la table qui associe à chaque appareil la mesure de la sonde multi-échelle, nous exprimons que C_k et C_l sont déployés respectivement sur D_i et D_j seulement si D_i et D_j ont la même valeur dans *MSProbe*, c'est-à-dire :

$$\forall i, j \in \{1, .., nb\_app\} \\ (SatVar(k,i) \wedge SatVar(l,j)) \implies (MSProbe(i) = MSProbe(j)) \quad (6.7)$$

Placement par instance d'échelle Enfin, la présence d'un composant à une certaine échelle (exprimée au moyen du mot-clé `Each`) est définie par une contrainte du même type que les précédentes, dans laquelle l'ensemble des appareils possibles est limité (et identifié à partir des valeurs mesurées par la sonde multi-échelle concernée). Par exemple, l'expression `Ck @ Each Some.MS.Scale` exprime qu'un composant C_k doit être déployé dans chaque instance d'échelle `Some.MS.Scale`. Pour cela, deux tableaux sont nécessaires : *MSProbe*, associant à chaque appareil la mesure de la sonde multi-échelle, et *MSProbeId*, listant les identifiants uniques de chaque instance d'échelle. L'expression précédente se traduirait par la contrainte (*nb_inst* étant le nombre d'instance d'échelle) :

$$\forall i \in \{1, .., nb\_inst\} \left(\sum_{\substack{j \in \{1,..,nb\_app\} \\ MSProbeId(i) = MSProbe(j)}} SatVar(k,j) \right) = 1 \quad (6.8)$$

6.3.3 Exemple

La table 6.1a est un exemple de la matrice *Comp* construite à partir de l'exemple présenté dans la section 5.2. La table 6.1b est un exemple de la matrice *Dom* extraite de la récupération d'un état du domaine. Nous considérons pour cet exemple un domaine contenant quinze appareils. Dans ces matrices, les propriétés sont :

- P_1 : `LinuxOrAndroid`,
- P_2 : `RAM80`,
- P_3 : `Screen`,
- P_4 : `CPURAM`,
- P_5 : `Administration.Level.Service("Toulouse.SharingBikes")`,
- P_6 : `Device.Type.Cloudlet`,
- P_7 : `SigFoxActive`,
- P_8 : `Device.Type.Smartphone`,

— P_9 : `User.NumberOfUsers.Group`,
— P_{10} : `Device.StorageCapacity.Giga`.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
Bike	1	0	0	0	1	1	0	0	0	0
BikeAvail	1	0	0	0	0	0	1	0	0	0
Stat	1	0	0	1	0	1	0	0	0	0
GUI	1	1	1	0	0	0	0	1	0	0
IM	1	1	0	0	0	0	0	1	1	0
IMHist	1	0	0	0	0	0	0	1	0	0
RP	1	0	0	0	0	0	0	0	0	1

(a) Matrice des composants *Comp*.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
A_1	0	1	0	1	0	1	1	0	0	0
A_2	1	1	1	1	0	0	0	1	1	0
A_3	1	1	1	1	0	0	0	1	1	0
A_4	1	1	1	1	1	1	1	0	0	0
A_5	1	1	1	1	0	0	0	1	1	0
A_6	1	1	1	1	0	0	0	1	1	0
A_7	1	1	1	1	0	0	0	1	1	0
A_8	1	1	1	0	0	0	0	1	0	0
A_9	1	1	0	1	1	0	0	0	0	1
A_{10}	1	1	0	1	1	0	0	0	0	1
A_{11}	1	1	0	1	1	1	1	0	0	0
A_{12}	1	1	0	1	1	1	1	0	0	0
A_{13}	1	1	1	1	1	1	1	0	0	0
A_{14}	1	1	0	1	1	1	1	0	0	0
A_{15}	1	1	0	1	1	1	1	0	0	0

(b) Matrice des appareils *Dom*.

Tableau 6.1 – Données des composants et des appareils.

Il faut noter que les sondes, y compris les sondes multi-échelles (en l'occurrence, pour notre exemple, les sondes `Administration` et `Device`), sont utilisées pour construire la matrice *Dom*. De manière générale, les mesures prises par les sondes sur les appareils font aussi partie de l'état du domaine. Elles sont fournies sous la forme d'une table associant appareil et mesure. Ainsi, pour notre exemple, la résolution nécessite des informations sur le service d'administration des appareils, leur type, leur capacité de calcul et l'appartenance à un groupe de personnes. C'est la sonde multi-échelle `Administration` qui détermine le service qui administre l'appareil, la sonde `Device` le type et la capacité de calcul de l'appareil et la sonde `Users` l'appartenance à un groupe. Les sondes `Users` et `Administration` produisent respectivement les tables 6.2a et 6.2b (ce sont des exemples). Dans ces tables, *n/a* signifie

que l'appareil n'appartient pas à l'échelle demandée ; il n'a donc pas de valeur d'instance d'échelle. Dans la table 6.2b, les abréviations Tlse.SB et Tlse.Metro font référence respectivement aux services Toulouse.SharingBikes et Toulouse.Metro.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
`Group`	*n/a*	GrA	GrA	*n/a*	GrB	GrB	GrA	GrC	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*

(a) Données sondées de `User`.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
`Service`	Tlse.Metro	*n/a*	*n/a*	Tlse.SB	*n/a*	*n/a*	*n/a*	Tlse.SB

	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
`Service`	Tlse.SB	Tlse.SB	Tlse.SB	Tlse.SB	Tlse.SB	Tlse.SB	Tlse.SB

(b) Données sondées de `Administration`.

Tableau 6.2 – Données des sondes multi-échelles.

Pour plus de clarté, nous avons modifié l'exemple en réduisant deux exigences du listing 5.5 :

— l'exigence d'intervalle (ligne 7) à `2..5` appareils au lieu de `10..30` ;
— l'exigence de ratio (ligne 6) à `1/3` au lieu de `1/5`.

La table 6.3 représente une matrice d'obligation possible c'est-à-dire un plan de déploiement, pour notre exemple de problème initialement défini dans le listing 5.5 (et dont le code complet se trouve dans l'annexe A) et les tableaux d'identification d'instances d'échelles constitués des mesures prises par les sondes `User.NumberOfUsers.Group` (table 6.2a) et `Administration.Level.Service` (table 6.2b).

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
Bike	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1
BikeAvail	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Stat	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
GUI	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0
IM	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0
IMHist	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
RP	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Tableau 6.3 – Matrice d'obligation *Oblig*.

6.4 Architecture en phase d'installation et d'activation

6.4.1 Installation

Une fois le plan de déploiement généré, le moniteur de déploiement lance sa réalisation. S'il y a eu une modification de l'état du domaine, une boucle autonomique doit prendre le relais pour adapter le plan de sorte qu'il satisfasse les propriétés initiales à préserver. Lors de cette phase, le système de déploiement utilise les sondes du composant des sondes spécifiques (version 1), ainsi que le composant Probes Supervision (V1) pour analyser les données obtenues et détecter l'insatisfaction de propriétés.

La figure 6.4 représente l'architecture du support local de déploiement à ce stade. Des agents supportent différentes activités (installation et activation du composant, supervision, etc.). Un composant Agent Platform est donc nécessaire pour supporter l'exécution de ces agents.

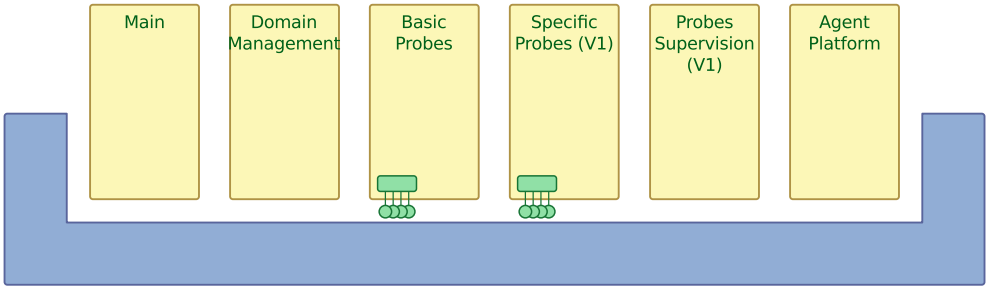


Figure 6.4 – Architecture du support local de déploiement avant l'installation

6.4.2 Activation

Une fois installés sur les différents appareils, les composants du système réparti multi-échelle sont activés (cf. figure 6.5).

À partir de là, les sondes définies dans le composant des sondes spécifiques (version 1) ne sont plus toutes nécessaires : seules celles concernant les propriétés à préserver à l'exécution sont à conserver. Ce composant est donc remplacé par un autre (version 2). De la même manière, une seconde version du composant de supervision remplace le premier.

La figure 6.6 représente la nouvelle architecture du support local de déploiement.

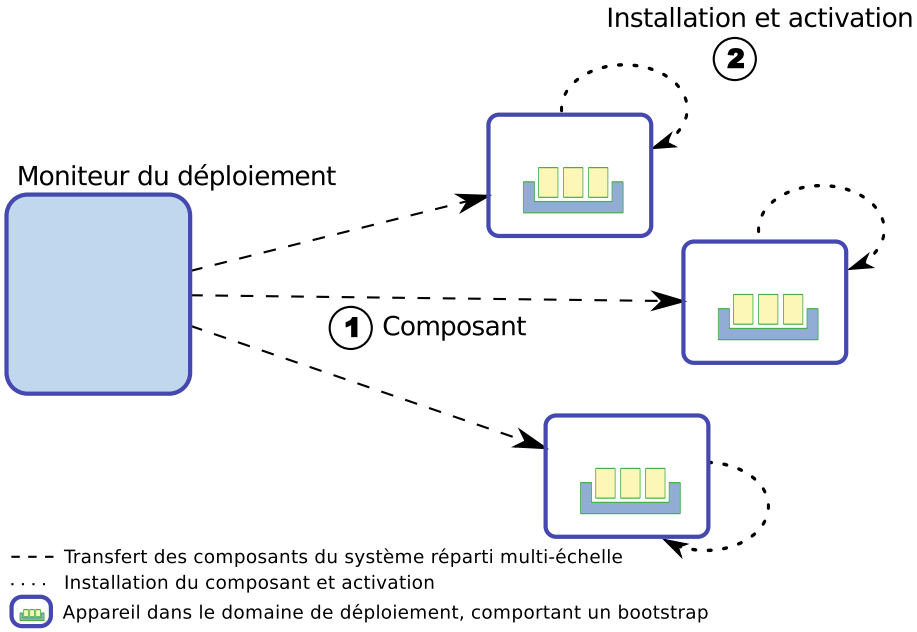


Figure 6.5 – Étapes de l'installation et l'activation

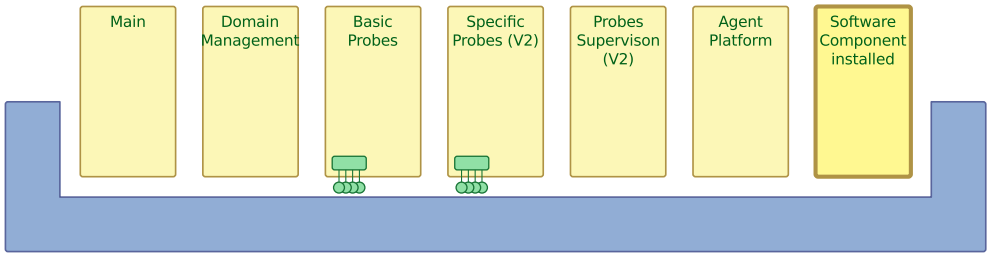


Figure 6.6 – Architecture du support local de déploiement après l'installation

6.4.3 Exécution

Une fois les composants du système déployé installés et activés, l'activité principale du système de déploiement est la supervision. Un nouveau composant est également installé (cf. figure 6.7), appelé Deployed Software Communication : il offre au système déployé un moyen d'interagir avec le support local de déploiement (demande de reconfiguration, échange d'informations, etc.).

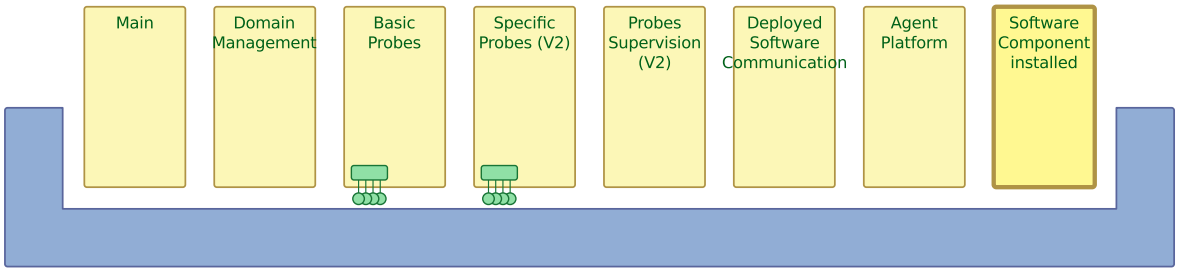


Figure 6.7 – Architecture du support local déploiement à l'exécution

6.5 Conclusion

Dans ce chapitre, nous avons proposé une architecture répartie qui supporte le déploiement initial de manière automatique, de la constitution du domaine et de l'acquisition de son état jusqu'à la réalisation du plan de déploiement. Cette phase est contrôlée par un élément centralisé, le moniteur de déploiement, qui s'appuie sur un gestionnaire du domaine et sur un support local de déploiement sur chaque appareil. Ce support local est un conteneur de composants, initialement minimal, qui s'enrichit dynamiquement de composants pour la réalisation du déploiement. En outre, il intègre une plateforme d'exécution pour des agents autonomes ; il est ainsi susceptible de supporter les adaptations autonomiques du plan de déploiement lors de l'exécution de l'application déployée. Par ailleurs, dans ce chapitre, nous avons montré comment les propriétés exprimées au moyen du langage MuScADeL peuvent être transformées afin de pouvoir être traitées par un solveur de contraintes. C'est ce solveur qui produit un plan de déploiement initial adapté à l'état du domaine et conforme aux spécifications exprimées.

Contribution

Afin de supporter le déploiement initial, une architecture répartie est proposée. Elle est composée d'un gestionnaire de domaine qui enregistre les entrées et les sorties des appareils, d'un moniteur de déploiement et d'un support local de déploiement sur chaque appareil. Le moniteur de déploiement utilise un solveur de contraintes pour produire le plan de déploiement. Pour cela, les propriétés spécifiées doivent être transformées en contraintes avant d'être traitées par le solveur. C'est aussi le moniteur de déploiement qui lance la réalisation du plan. Initialement, le support local de déploiement est un bootstrap minimal. Il s'enrichit de nouveaux composants au fur et à mesure de l'avancée du processus, et héberge à la fois les composants du système de déploiement et ceux de l'application déployée.

7 Déploiement autonomique : architecture et situations d'adaptation

Problématique

Le déploiement initial effectué, le système de déploiement doit maintenir en condition opérationnelle le système déployé. Le domaine de déploiement étant dynamique, des perturbations vont intervenir lors de l'exécution du système déployé. Le système de déploiement doit pouvoir identifier ces situations, les analyser et exécuter la solution si elle existe. De plus, la taille du domaine de déploiement ne permet pas une gestion centralisée de cette dynamique. Le système de déploiement doit pouvoir prendre en compte ces situations de manière décentralisée.

7.1 Introduction

7.1.1 Dynamique du domaine de déploiement

Le déploiement d'un système réparti multi-échelle s'effectue en deux étapes : le déploiement initial (cf. figure 4.9), que nous avons exposé dans les chapitres précédents, et le maintien en condition opérationnelle à l'exécution du système. Ceci s'effectue en faisant évoluer le plan de déploiement en fonction de la dynamique du domaine de déploiement.

De par la nature du domaine, le système déployé va subir des perturbations. Nous dénotons trois types de perturbations :

D'une part, l'apparition de nouveaux appareils. Ce cas de figure aura été pris en compte dès la spécification du déploiement par le concepteur qui aura défini des propriétés dynamiques sur certains composants. Le système de déploiement doit pouvoir intégrer ces nouveaux appareils dans le plan de déploiement.

Puis, la variation de l'état du domaine menant à une inconsistance (c'est-à-dire propriétés de déploiement insatisfaites). Ce sont les changements de l'état d'un appareil qui

viole une proriété du composant hébergé ou une exigence de ce composant : la mémoire vive n'est plus suffisante, le réseau exigé n'est plus actif, etc. Le système de déploiement doit être à même de faire évoluer le plan de déploiement en trouvant une solution, si elle existe, à ces violations de propriétés.

Enfin, la disparition d'un appareil hébergeant un composant n'ayant pas de propriété dynamique spécifiée : un composant qui doit être déployé sur dix appareils exactement, etc. Le système de déploiement doit déployer à nouveau ce composant sur un appareil qui peut l'héberger.

Afin de répondre à cette dynamique, une adaptation doit pouvoir être effectuée au plus près des appareils ciblés.

7.1.2 Déploiement autonomique

La taille du domaine de déploiement influe sur la manière dont est conçu le système de déploiement. Si le système de déploiement était centralisé, cela aurait pour conséquences une latence dans les temps de réponse, une utilisation massive de la bande passante, et des risques de surcharge du moniteur de déploiement. L'hétérogénéité des appareils impliqués dans le déploiement induit le besoin d'une réponse adaptée aux appareils dans le domaine de déploiement et aux composants déployés en cas de perturbations. De ce fait, afin d'être capable de répondre efficacement aux perturbations, le système de déploiement doit être décentralisé.

Le système de déploiement doit donc avoir deux caractéristiques principales pour la gestion du déploiement à l'exécution de l'application. D'un côté, il faut qu'il soit décentralisé afin de pouvoir réactif, en ayant une réponse rapide et locale à la dynamique du domaine de déploiement. D'un autre côté, il doit être autonome afin de pouvoir trouver une solution adaptée aux perturbations rencontrées au cours de l'exécution du système déployé.

En 2001, IBM a soulevé dans un manifeste [Horn, 2001] le problème que poserait la complexité des systèmes informatiques à venir. Les systèmes devenant de plus en plus complexes, grands en taille et distribués, la gestion et l'administration par l'humain de tels systèmes s'avère une tâche colossale. La réponse à ce problème a été l' « informatique autonomique » (*autonomic computing*).

L'informatique autonomique désigne un ensemble de solution pour qu'un système informatique se gère lui-même à partir de spécifications et d'objectifs de haut-niveau donnés par un administrateur [Kephart and Chess, 2003]. L'autonomie de ces systèmes se base sur quatre principes : l'auto-configuration, l'auto-optimisation, l'auto-réparation et l'auto-protection. Afin de pouvoir réaliser ces tâches d'auto-gestion, le système autonomique a besoin d'avoir des informations sur l'état du système, analyser ces données, prendre une décision, et l'appliquer. Ce processus est défini pas la boucle autonomique introduite par Kephart et Chess dans [Kephart and Chess, 2003]. La figure 7.1 présente cette boucle de l'informatique autonomique. Dans un premier temps, le système perçoit un changement dans l'environnement (*Monitor*), et analyse ce changement en prenant en compte les informations disponibles (*Analyze*, *Knowledge*). Puis, l'action à appliquer sera planifiée (*Plan*) pour une exécution immédiate ou différée de la solution (*Execute*).

L'informatique autonomique est indiquée comme solution à notre problème de maintien en condition opérationnelle de l'application pour répondre aux besoins d'adaptation décentralisée. La figure 7.2 illustre les mécanismes autonomiques décentralisés requis pour

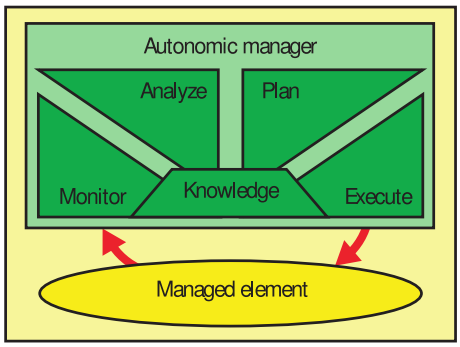


Figure 7.1 – Boucle autonomique : perception, analyse, planification et exécution [Kephart and Chess, 2003]

le système de déploiement.

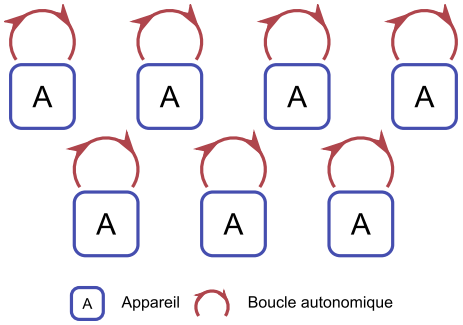


Figure 7.2 – Mécanismes autonomiques décentralisés

Dans cette thèse, nous nous sommes concentrés sur les aspects processus, spécification du déploiement avec le langage dédié et déploiement initial. Dans ce chapitre, nous commençons par analyser les situations d'adaptation qui peuvent intervenir à l'exécution du système déployé (section 7.2). Puis, nous apportons quelques éléments de solutions à ces situations d'adaptation, dans le cadre d'une architecture du système de déploiement (section 7.3).

7.2 Les situations d'adaptation

Dans cette section, nous présentons les situations d'adaptation de déploiement autonomique. Dans un premier temps, nous proposons une catégorisation des situations. Puis suivant les phases de la boucle autonomique, nous décrivons la perception de ces situations, leur analyse et leur planification.

7.2.1 Inventaire

Nous choisissons de classer les situations d'adaptation en cinq catégories : celles correspondant à la dynamique du domaine, la dynamique de l'état du domaine, la dynamique de l'application, les pannes logicielles de l'application, et les pannes du système de déploiement.

Les situations d'adaptation correspondant à la **dynamique du domaine** sont celles qui prennent en compte l'apparition et la disparition d'appareils. Ces situations peuvent être de deux sortes : celles traitant des propriétés dynamiques sur les composants et les autres. Lors de la spécification du déploiement, le concepteur prend en compte la dynamique du domaine en définissant des propriétés dynamique sur certains composants (cf. section 5.3.6). Par exemple, en utilisant le mot-clé `All`, le concepteur prend en compte le fait que des appareils entrent dans le domaine de déploiement, et il définit comme propriété : pour chaque appareil entrant, si toutes les propriétés du composant concerné sont satisfaites, ce composant doit y être déployé. Le système de déploiement doit alors prendre en compte l'apparition des nouveaux appareils et y déployer les composants concernés. Si une propriété dynamique n'est pas définie, la disparition d'un appareil conduit à une violation des propriétés de déploiement. Le système de déploiement doit alors arriver à une solution satisfaisant la spécification du concepteur.

La **dynamique de l'état du domaine** conduit a des situations de violation des propriétés de déploiement. Le plan de déploiement est ainsi dans un état non conforme aux spécifications. Le système de déploiement doit aboutir à une solution adaptée en fonction du plan de déploiement actuel, des propriétés à préserver et de l'état du domaine, si elle existe. Les situations dans cette catégorie sont celle provenant d'une évolution des caractéristiques de l'appareil menant à une violation des propriétés de déploiement.

La **dynamique de l'application** conduit aussi à des situations d'adaptation. À l'exécution du système déployé, l'opérateur humain de déploiement peut ajouter ou retirer des composants. Le système de déploiement doit ainsi prendre en compte ces nouveaux composants avec leurs propriétés propres et les intégrer au système déjà déployé.

Certaines situations peuvent provenir de **pannes logicielles de l'application**. Ces pannes peuvent survenir à l'installation du composant, être une indisponibilité de composant à déployer (adresse injoignable, erreur de téléchargement, téléchargement corrompu).

Les **pannes du système de déploiement** lui-même sont aussi sujettes à des situations d'adaptation. La plateforme du modèle de composant ou un des composants formant le support local du système de déploiement peut s'arrêter brutalement. Le gestionnaire de déploiement peut aussi devenir inaccessible (serveur distant indisponible) ou le réseau local tomber en panne (communication pair à pair indisponible).

7.2.2 Perception

Toutes ces situations d'adaptation peuvent être perçues par le système de déploiement lui-même. Nous considérons différentes méthodes de perception de situations d'adaptation :

- la détection de panne : dégradation de l'état d'un appareil, disparition d'appareil, etc.
- la réception d'une annonce du système de déploiement : enregistrement d'un nouvel appareil, disparition annoncée d'un appareil, etc.
- la réception d'une information d'erreur : erreur de téléchargement, erreur d'installation, erreur de communication, etc.

La table 7.1 regroupe les différentes situations d'adaptation et leur moyen d'identification. Les abréviations Dyn.Dom., Dyn.Ét., Dyn.App., Pan.App. et Pan.Dep. font référence respectivement aux catégories dynamique de domaine, dynamique de l'état du domaine, dynamique de l'application, pannes logicielles de l'application et pannes du système de déploiement. L'identification des situations est relative à la catégorie. Les situations provenant de la dynamique du domaine (hormis la disparition constatée d'un appareil) et de

la dynamique de l'application sont identifiées *via* une réception d'annonce. La disparition constatée d'un appareil est identifiée *via* une détection d'une panne car elle mène à une violation des propriétés de déploiement, comme les situations de la catégorie dynamique de l'état du domaine. Les situations des catégories panne logicielle de l'application et panne du système de déploiement sont quant à elles identifiées *via* la réception d'un erreur.

Catégorie	Situation	Identification
Dyn.Dom.	Disparition annoncée	Réception d'une annonce
Dyn.Dom.	Disparition constatée a priori	Détection d'une panne
Dyn.Dom.	Apparition annoncée	Réception d'une annonce
Dyn.Dom.	Apparition constatée a priori	Réception d'une annonce
Dyn.Ét.	Dégradation de l'état	Détection d'une panne
Dyn.App.	Apparition d'un composant	Réception d'une annonce
Pan.App.	Panne du composant	Réception d'une erreur
Pan.Dep.	Panne du système de déploiement	Réception d'une erreur
Pan.Dep.	Panne réseau local	Réception d'une erreur
Pan.App.	Disponibilité du composant à déployer	Réception d'une erreur
Pan.App.	Problème de sécurité	Réception d'une erreur
Pan.Dep.	Panne du gestionnaire de domaine	Réception d'une erreur

Tableau 7.1 – Situations d'adaptation et leur identification

7.2.3 Analyse

Une première analyse des différentes situations permet de les réduire en deux situations simples principales : apparition ou disparition d'appareil. La troisième situation simple et l'apparition d'un composant. La table 7.2 présente cette correspondance.

Lors d'une disparition constatée d'un appareil ou d'un panne du système déployé, le système de déploiement va réagir considérant ces cas comme des disparitions d'appareil, le composant déployé n'est plus actif au sein du domaine de déploiement. La « disparition contrôlée » est une disparition d'appareil qui est prise en compte par le support local du système déploiement. Elle est utilisée pour certains où un traitement supplémentaire peut-être effectué : par exemple pour la disparition annoncée d'un appareil, les composants hébergés vont être préalablement désactivés. Il est en de même pour la panne locale ou la panne du réseau local : l'appareil disparaît du domaine de déploiement, prenant connaissance de cela, il désactive le composant qui n'est plus utile actif sur l'appareil.

L'apparition d'un composant se traite différemment des autres situations, car la source de la perturbation est un nouvel élément à prendre en compte, et non une adaptation des composants déjà déployés.

Cette phase de simplification permet d'affiner notre analyse des situations d'adaptations, ce qui permettra une planification plus claire et plus concise des solutions.

Situation	Situation simple
Disparition annoncée	Disparition contrôlée de l'appareil
Disparition constatée a priori	Disparition de l'appareil
Apparition annoncée	Apparition de l'appareil
Apparition constatée a priori	Apparition de l'appareil
Dégradation de l'état	Disparition contrôlée de l'appareil
Apparition d'un composant	Apparition du composant
Panne du composant	Disparition contrôlée de l'appareil
Panne du système de déploiement	Disparition de l'appareil
Panne réseau local	Disparition contrôlée de l'appareil
Disponibilité du composant à déployer	Disparition de l'appareil
Problème de sécurité	Disparition de l'appareil
Panne du gestionnaire du domaine	Disparition de l'appareil

Tableau 7.2 – Simplification des situations d'adaptation

7.2.4 Planification

La planification des situations simples s'effectue de différentes manières.

La situation d'apparition d'un composant se distingue de la disparition ou l'apparition d'un appareil. La planification de cette situation doit prendre en compte des éléments spécifiques. En effet, l'apparition d'un composant conduit au déploiement du(des) composant(s) sur les appareils du domaine, en fonction des propriétés de déploiement définies sur ce(s) composant(s).

La décision concernant l'action à mener dans le cas des situations simples de disparition et d'apparition d'appareil est prise en fonction de l'état du domaine et des spécifications des propriétés de déploiement. Si le concepteur du déploiement a spécifié des propriétés dynamiques, le système de déploiement doit déployer le(s) composant(s) sur lesquels portent ces propriétés sur les appareils vierges entrant dans le domaine. De la même manière, si un appareil disparaît et qu'une propriété fixe le nombre de composants déployés, le système de déploiement doit réagir en déployant ce composant sur un autre appareil qui satisfasse les propriétés du composant concerné. La table 7.3 présente les différentes actions du système de déploiement, en fonction de la situation simple. Une différenciation est faite entre un appareil entrant vierge et non vierge – un appareil non vierge étant un appareil ayant le composant concerné installé et désactivé. En effet, il y a une différence d'action en fonction de la présence ou non du composant concerné sur l'appareil. Dans cette table, nous supposons que l'appareil apparaissant est compatible avec le composant concerné (contraintes et exigences satisfaites). Les symboles ↷, ▷, ↻, ↰ et ø représentent respectivement les actions de déploiement, d'activation, de redéploiement (déployer le composant sur un autre appareil du domaine), désinstallation, et aucune action à effectuer.

La propriété de l'intervalle spécifie qu'un type de composant doit avoir entre *min* et *max*

Spécification et état du domaine		Disparition d'un appareil	Apparition d'un appareil vierge	Apparition d'un appareil non vierge
Simple (nombre exact d'instances)		↺	ø	ø
Intervalle	$<$ min	↺	*n/a*	*n/a*
	$>$ min et $<$ max	ø	↷ ?	▷ ?
	$>$ max	*n/a*	ø	ø
`All`		ø	↷	▷
`Each`	n^{elle} instance d'échelle	*n/a*	↷	▷
	instance d'échelle existante	*n/a*	ø	ø
	disparition instance d'échelle	ø	*n/a*	*n/a*
Ratio - x/y	x++	*n/a*	ø	ø
	x- -	↺ x	*n/a*	*n/a*
	y++ $y^n[y^r]$=0	*n/a*	↷ x ↷ y	↷ x ▷ y
	y++ $y^n[y^r]$!=0	*n/a*	ø	ø
	y- - $y^n[y^r]$=0	↰ x	*n/a*	*n/a*
	y- - $y^n[y^r]$!=0	ø	*n/a*	*n/a*

Tableau 7.3 – Actions des situations simples : apparition et disparition d'appareil

instances déployées dans le domaine. Le cas de l'intervalle se subdivise en trois sous-cas : si le nombre de composants déployés est inférieur à la valeur minimale, entre les valeurs minimale et maximale, et supérieur à la valeur maximale. Nous choisissons de ne pas déployer de composants supplémentaires si le nombre de composants déployés est dans l'intervalle.

La propriété `Each` concerne le déploiement d'un composant par instance d'échelle. Pour ce cas, trois sous-cas sont identifiés : si l'apparition d'un appareil crée une nouvelle instance d'échelle, si l'instance d'échelle est déjà existante, et si la disparition d'un appareil supprime l'instance d'échelle. Ce n'est que dans le premier sous-cas qu'une action est requise, car un composant doit être déployé dans cette nouvelle instance. Pour la disparition d'instance d'échelle, le composant déployé disparaît avec l'appareil, donc aucune action n'est requise.

La propriété de ratio spécifie le déploiement d'un composant x en fonction du nombre de composants y déployés, et respectant une expression de ratio x^r/y^r. Nous noterons y^n le nombre total n de composants y déployés. Six sous-cas peuvent survenir :

— la disparition d'un appareil hébergeant un composant x,
— l'apparition d'un appareil pouvant héberger un composant x,
— l'apparition d'un appareil pouvant héberger un composant y avec y^n divisible par y^r,
— l'apparition d'un appareil pouvant héberger un composant y avec y^n non divisible

par y^r,
— la disparition d'un appareil hébergeant un composant y avec y^n divisible par y^r,
— la disparition d'un appareil hébergeant un composant y avec y^n non divisible par y^r.

Dans ces quatre derniers cas, l'action sur un composant y peut mener à une action sur le composant x. Si l'installation ou la désinstallation (disparition) d'un composant y correspond aux valeurs de l'expression du ratio, il faut aussi installer un composant x sur un appareil du domaine de déploiement ou en désinstaller un.

7.3 Éléments de solution

Dans cette section, les principales idées de réalisation sont décrites. Nous présentons les choix technologiques que nous avons faits, puis l'architecture générale du système de déploiement, et enfin l'étude d'un cas particulier de l'architecture que nous avons proposée.

7.3.1 Choix technologiques

Afin d'effectuer cette adaptation autonomique à la dynamique du domaine et aux variations de l'état du domaine de déploiement, nous avons effectué certains choix technologiques.

7.3.1.1 Décentralisation et autonomie

Pour assurer une analyse et une prise de décision locale, nous proposons d'utiliser des agents [Ferber, 1995]. Les agents sont des entités autonomes, décentralisés qui sont adaptés à l'auto-configuration et l'auto-réparation locales requises. Nous proposons une supervision hiérarchique des appareils. Afin de déterminer l'organisation de cette supervision, nous nous basons sur une idée proche du multi-échelle : la supervision hiérarchique avec un superviseur par instance d'échelle. Les échelles étant ordonnées et ayant une construction hiérarchique, elles sont adaptés à ce type de supervision. La figure 7.3 représente cette supervision hiérarchique par échelle.

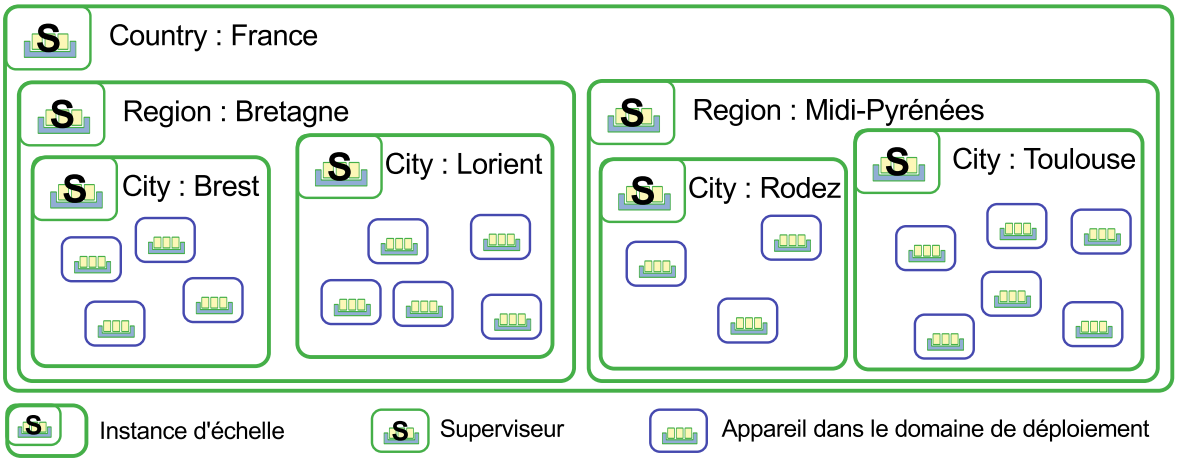


Figure 7.3 – Supervision hiérarchique par échelle : point de vue Geography et dimension Location

Dans la dimension Geography.Location, les échelles City, Region et Country sont incluses l'une dans l'autre. Cela nous permet d'avoir différents superviseurs en fonction des instances de l'échelle la plus petite, City, avec les instances d'échelles Brest et Lorient, et Rodez et Toulouse. Les instances de l'échelle Region, Bretagne et Midi-Pyrénées, sont leurs superviseurs hiérarchiques respectifs. De même, l'échelle Country, dont l'instance est France, est le superviseur hiérarchique des instances de l'échelle Region.

Si un appareil a pour plus petite échelle Region (et non City) et est dans l'instance d'échelle Bretagne, il a comme superviseur celui de cette instance d'échelle.

7.3.1.2 Gestion du domaine

Le gestionnaire de domaine de déploiement est dans notre cas un ensemble d'instances de serveurs RabbitMQ [Rab] (protocole AMQP [AMQ]) et un serveur agrégeant le liste des appareils disponibles. L'utilisation de RabbitMQ permet une prise en charge des appareils répondant aux problématiques de l'hétérogénéité des réseaux, de passage de firewall, etc. Les instances de serveurs RabbitMQ permettent aussi de prendre en compte le passage à l'échelle du domaine de déploiement grâce à leur capacité à créer de nouvelles instances pour tenir la charge face à la demande. De plus, l'asynchronisme du protocole évite les attentes actives de réponses, par exemple en cas de disparition d'appareil. Pour détecter la disparition, un signal périodique *heartbeat* (battement de cœur) permet de s'assurer de la présence des appareils dans le domaine de déploiement.

Initialement, il était prévu d'utiliser le protocole distribué XMPP [XMP] pour détecter la présence d'appareils ainsi que pour supporter la communication entre eux. Pour cela, chaque client doit avoir un compte sur l'un des serveurs Jabber existant (les serveurs sont organisés en une « fédération » de serveurs, ce qui assure le passage à l'échelle du protocole de présence). Ceci implique que les utilisateurs ouvrent un compte, ce qui ne semble pas toujours pratique, et surtout pas automatisable aujourd'hui sur les serveurs disponibles. Une solution alternative serait de fournir notre propre serveur Jabber autorisant les créations de compte dynamiques. Toutefois, même si cette solution est simple (elle a été également testée), elle fait perdre toute la puissance de la fédération de serveurs puisque tous les clients se connectent à un serveur unique, donc sans passage à l'échelle possible. De plus, il n'est pas simple avec XMPP d'échanger des messages de manière sûre et les échanges, au format XML, sont peu performants. Aussi, le choix de cette technologie, qui était initialement justifié par son protocole de présence simple à mettre en œuvre, a été reconsidéré.

Les besoins sont les suivants :
— échanger des messages entre un appareil « maître » (pilotant le déploiement) et des appareils clients répartis à l'échelle d'Internet (donc dans des sous-réseaux, etc.),
— disposer d'un protocole de présence entre ces mêmes appareils,
— communiquer de manière fiable, et passant au travers de firewalls/routeurs (avec du NAT, *Network Address Translation*, ce qui exclut le pair à pair),
— assurer le passage à l'échelle du « système ».

À l'issue d'une phase de veille technologique, nous avons donc choisi le protocole AMQP [AMQ] et son implémentation RabbitMQ [Rab].

L'appareil dispose de plusieurs niveaux d'interaction. Il peut communiquer directement avec le gestionnaire de domaine de déploiement, avec son superviseur hiérarchique ou un autre appareil. L'apparition d'un appareil dans le domaine de déploiement peut se réaliser de deux manières : soit par la notification de présence de l'appareil à un su-

perviseur hiérarchique, soit par l'enregistrement de l'appareil dans le gestionnaire de domaine de déploiement. La technologie agent permet une communication pair à pair entre agents sur différents appareils. De plus, nous imaginons aussi une communication entre appareils transparente entre les appareils (hôte ou superviseur) en utilisant un protocole de découverte de service, comme le protocole UPnP [UPn].

7.3.2 Architecture générale

Le système de déploiement est composé de quatre éléments principaux : le moniteur de déploiement, le gestionnaire de domaine, les superviseurs hiérarchiques, et les agents locaux du support local du système de déploiement. La figure 7.4 représente cette architecture.

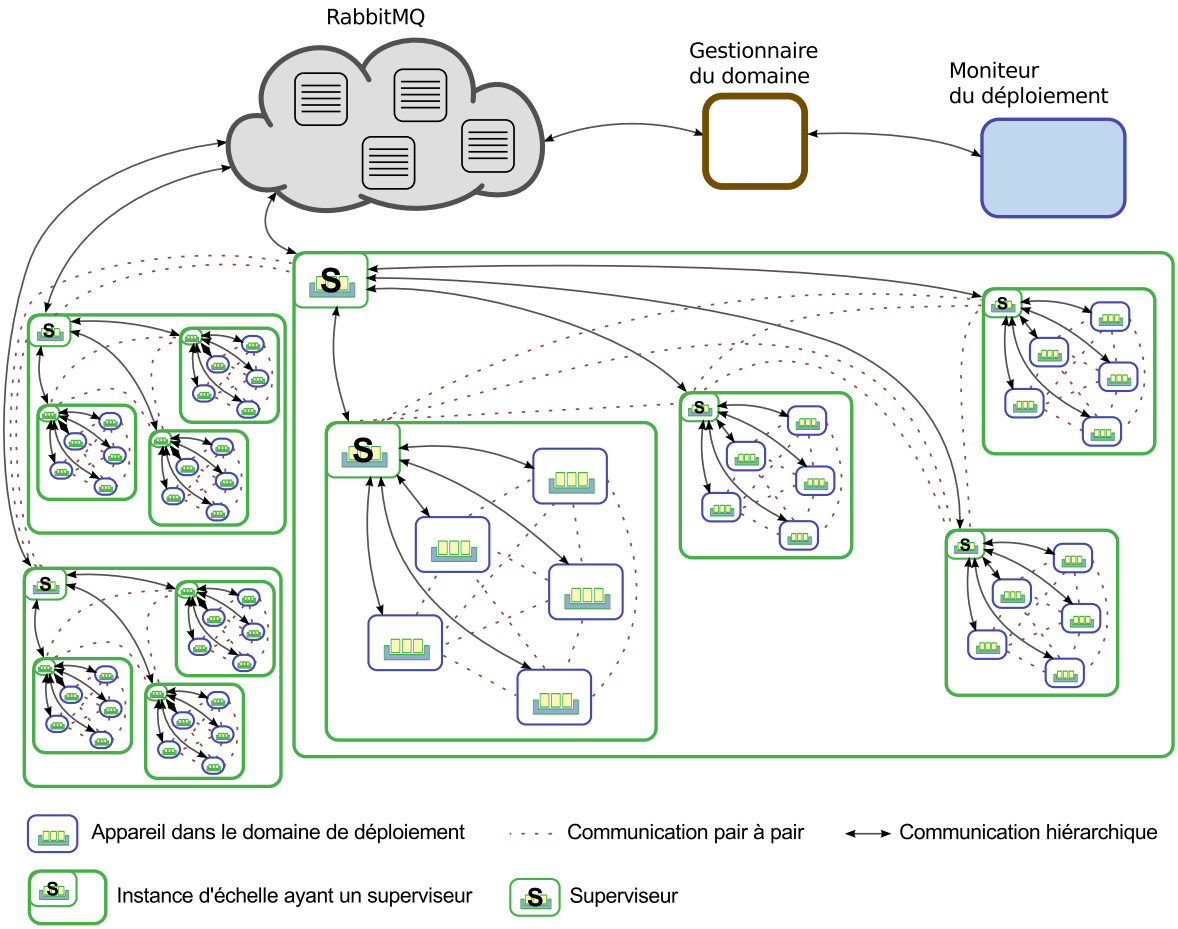


Figure 7.4 – Architecture générale du système de déploiement

L'identification de situation peut se faire de manière locale. La détection de violation de propriétés de déploiement se fait par l'agent localement sur l'appareil. Les sondes présentes sur les appareils permettent une supervision des paramètres significatifs des composants hébergés. Cette supervision est effectuée par un agent qui, pour des valeurs changeantes (par ex. la charge du processeur, la mémoire vive, etc.), va demander l'information périodiquement afin de la comparer avec la valeur spécifiée par le concepteur. Si la valeur réelle ne correspond plus aux spécifications, le thread diffuse localement l'information avec un message de violation de contraintes.

Le gestionnaire de domaine de déploiement est centralisé. Il utilise les différentes instances des serveurs RabbitMQ pour le passage à l'échelle. En effet, un des point fort de RabbitMQ est l'ajout à chaud de nouvelles instances pour tenir la charge. Cette centralisation est nécessaire pour la phase de déploiement initial, pour l'enregistrement des appareils et la récupération de l'état du domaine afin de générer un plan de déploiement. À l'exécution de l'application, pour le maintien du plan de déploiement, il est possible de passer à une architecture décentralisée de la gestion du domaine. Nous pouvons utiliser le système de filtre de RabbitMQ afin d'avoir une gestion du domaine par instance d'échelle. Les messages concernant les appareils peuvent être étiquetés en fonction de l'instance d'échelle dans laquelle ils sont et les superviseurs s'abonner à la file de message qui correspond à leur instance d'échelle.

7.3.2.1 Exécution

La décision d'effectuer ou non un nouveau déploiement d'un composant peut se prendre localement, et son application déléguée au superviseur hiérarchique. En identifiant et analysant la situation, l'agent peut agir localement sur le déploiement du composant, en utilisant les commandes du framework du modèle de composant. Si l'action requise concerne un autre appareil, il communique à l'agent présent sur le superviseur l'action à mener en utilisant la communication pair à pair fournie le système d'agent. L'agent sur le superviseur peut ainsi trouver un appareil compatible avec le composant à redéployer, et contacter l'agent d'installation sur cet appareil. Les agents présents sur cet appareil utilisent le framework du modèle de composant afin d'effectuer le déploiement local. Si le superviseur hiérarchique ne peut appliquer la décision, il la fait remonter à son propre superviseur, etc. Si le plus haut superviseur hiérarchique ne trouve pas d'appareil compatible, l'information remonte au moniteur de déploiement, qui en informe l'opérateur de déploiement. Le superviseur communique avec le moniteur de déploiement en passant par une instance de serveur RabbitMQ.

7.3.3 Conflit du contrôle par instance d'échelle

Avec la supervision hiérarchique par instance d'échelle, des cas de conflits peuvent se produire lors de la prise de décision. Si un appareil est situé dans deux instances d'échelles différentes, le choix du contrôleur hiérarchique qui doit gérer cette disparition se pose.

La figure 7.5 illustre ce cas au moyen d'un exemple. Nous avons deux dimensions, Network.Type et Geography.Location, et dans chacune de ces dimensions une échelle, respectivement LAN et City. Nous nous intéressons aux instances d'échelle persée pour LAN et Toulouse pour City. Le concepteur de déploiement a défini trois exigences décrite dans le listing 7.1.

Ces trois exigences spécifient respectivement que le composant `Cp` doit avoir 3 instances déployées dans le réseau local persée, que le composant `Ct` doit avoir 4 instances déployées dans la ville de Toulouse, et que le composant `Ctp` doit être déployé sur un appareil sur le réseau local que `Cp` et dans la même ville que `Ct`. Comme nous pouvons le voir dans la figure, l'appareil Atp appartient au réseau local persée et est dans la ville de Toulouse. Si cet appareil disparaît se pose le problème de la conflit du contrôle. Si le contrôleur `C_Ct` prend en charge l'adaptation autonomique, il risque de déployer une instance du composant `Ct` sur un appareil qui n'est pas dans le réseau local persée, et inversement si le contrôleur `C_Cp` prend la décision (pour le composant `Cp`).

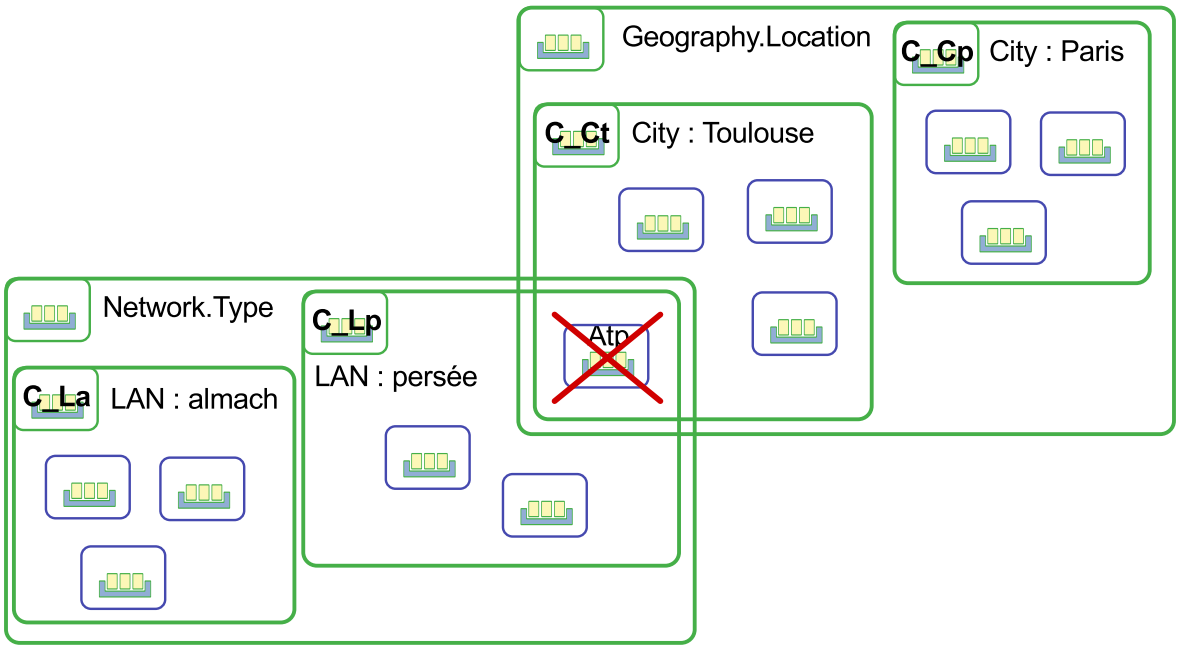


Figure 7.5 – Conflit dans le cadre de contrôle hiérarchique par instance d'échelle

```
1 Deployment {
2     ...
3     Cp @ 3, Network.Type.LAN("persee");
4     Ct @ 4, Geography.Location.City("Toulouse");
5     Ctp @ SameValue Network.Type.LAN(Cp),
6             SameValue Geography.Location.City(Ct);
7 }
```

Listing 7.1 – Exigence de déploiement illustrant le conflit du contrôle par instance d'échelle

Afin de résoudre ce conflit, les deux contrôleurs en conflit effectuent une négociation afin de trouver une solution. S'ils échouent, l'information remonte aux contrôleurs directs qui effectuent à leur tour des négociations. Si aucune solution n'est trouvée au terme de la négociation entre les deux plus hauts contrôleurs, l'opérateur humain de déploiement est informé de la non conformité aux spécifications de déploiement.

7.4 Conclusion

Nous avons présenté dans ce chapitre des éléments pour la prise en charge du déploiement autonomique. Une fois le plan de déploiement généré et le déploiement initial effectué, le système de déploiement doit maintenir un plan de déploiement, en conformité avec les spécifications du concepteur. L'adaptation en réponse à l'évolution de l'état du domaine se fait de deux manières, en réponse au dynamisme du domaine : soit réagir de manière prévue dans la spécification du déploiement (dynamisme prévu par le concepteur), soit réagir face à un évènement « imprévu » et induisant une violation des propriétés du déploiement. Dans ce dernier cas, la boucle autonomique répond à ce problème. Les agents présents dans les appareils peuvent gérer localement une dégradation de l'état de l'appa-

reil. Le système de déploiement étant décentralisé, il repose sur un système hiérarchique basé sur les instances d'échelles pour la gestion des situations d'adaptations complexes. Si localement le problème ne peut être résolu, l'agent transfère les données requises au superviseur hiérarchique pour déléguer la prise de décision ou son application.

Dans cette thèse, nous nous sommes focalisés sur le processus, la spécification du déploiement et le déploiement initial des systèmes répartis multi-échelles. Nous n'avons pas été jusqu'à la mise en oeuvre complète du déploiement autonomique. Toutefois, des choix technologiques permettant la mise en place du système autonomique ont été fait et certaines situations d'adaptation ont été implémentées, comme c'est le cas pour le déploiement continu, lors de l'apparition d'un appareil.

Une piste pour la gestion du déploiement autonomique décentralisé est l'utilisation d'agents pour la prise de décision locale et sa réalisation. L'organisation du système de déploiement est propice à l'utilisation des systèmes multi-agents [Ferber, 1995]. Les différentes situations d'adaptation identifiées peuvent servir de base pour de futurs projets de recherche qui se concentreront sur la partie autonomique du système de déploiement.

Contribution

Afin de pouvoir répondre aux perturbations lors de l'exécution de l'application, le système de déploiement réagit de manière autonomique. Nous avons identifié les différentes situations d'adaptations, les avons analysées. Puis, une solution est proposée pour chaque situation, en fonction de l'état du domaine et les spécifications du concepteur du déploiement (indication de dynamique, violation de propriétés). Au vu des conditions à l'exécution du système déployé (dynamique et taille du domaine), une architecture centralisée n'est pas envisageable. Une architecture décentralisée basée sur une supervision par instance d'échelle est proposée. Elle permet une réponse autonomique locale aux situations d'adaptations rencontrées.

8 Implémentation et validation

Problématique

Pour mettre en œuvre notre solution de déploiement sous la forme d'un prototype, des choix technologiques doivent être faits. Ils concernent l'outillage du DSL MuScADeL, la satisfaction des contraintes, la gestion du domaine, et plus généralement le middleware de déploiement. Le prototype doit également être évalué au regard des exigences, de manière qualitative et quantitative, afin de fournir des éléments de validation de notre proposition.

8.1 Introduction

Dans ce chapitre nous présentons notre réalisation de la solution pour le déploiement de systèmes répartis multi-échelles : il s'agit de MuScADeL et de l'outillage associé et du middleware MuScADeM. À partir d'un descripteur MuScADeL, MuScADeM permet la récupération de l'état du domaine de déploiement, la génération d'un plan de déploiement, sa réalisation, et le maintien en condition opérationnelle de l'application. MuScADeM contient une partie centralisée, le moniteur de déploiement (là où s'effectue la génération du plan de déploiement), et une partie décentralisée, le support local du système de déploiement sur les appareils.

Ce chapitre s'organise comme suit. La section 8.2 présente les différents éléments du middleware MuScADeM. Dans cette section, nous faisons aussi un parallèle avec les exigences de la solution de déploiement. Puis, nous présentons les différentes démonstrations que nous avons eu l'opportunité de présenter au cours de la thèse à la section 8.3. Enfin, dans la section 8.4, nous discutons des performances de notre solution en matière de génération du plan de déploiement initial et de passage à l'échelle.

8.2 Réalisations

Dans cette partie, nous décrivons différents éléments du système de déploiement que nous avons implémentés. À l'annexe C sont regroupées des exigences de la solution pour le

déploiement de systèmes répartis multi-échelles, issues de l'analyse réalisée dans le cadre du projet INCOME. Tout au cours de cette section, des annotations dans la marge permettent de préciser quelles exigences sont satisfaites par la réalisation présentée.

EX43

EX44

EX45

Le processus de déploiement initial du système réparti multi-échelle présenté dans la figure 8.1 est une extension de la figure 4.9, dans laquelle nous avons précisé les outils technologiques utilisés.

Nous suivons l'ordre chronologique de ce processus pour présenter nos réalisations et choix technologiques.

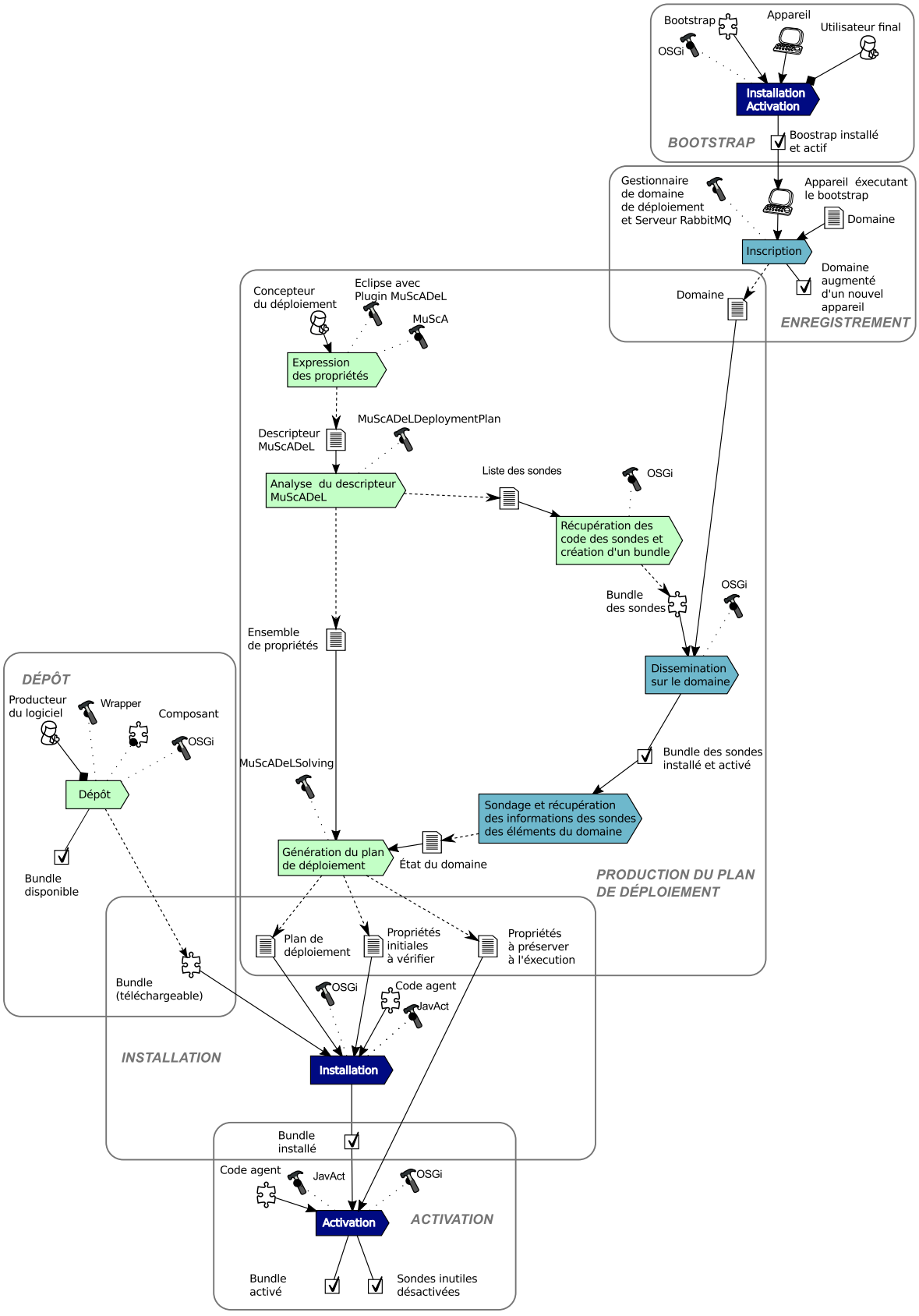


Figure 8.1 – Processus de déploiement initial avec les choix technologiques

8.2.1 Bootstrap

EX14

Le bootstrap est un système d'amorce installé sur l'appareil, sur lequel s'appuie le système de déploiement . C'est un conteneur OSGI contenant différents bundles de base – le bundle est l'unité de déploiement du framework OSGI. Par la suite, le support local du système de déploiement va s'enrichir en ajoutant des bundles à ce bootstrap. L'implémentation d'OSGi utilisée est Apache Felix [Fel], version 4.2.1. Ce choix a été fait d'une part car cette implémentation est open source, et d'autre part parce qu'elle est facilement déployable sur des dispositifs de petite taille, tels des smartphones sous Android.

En pratique, pour des raisons d'hétérogénéité, il existe deux versions du bootstrap : une pour Java (J2SE standard) et une pour Android (Dalvik). Les différences entre ces deux versions résident dans l'implémentation du code de lancement de Felix, soit destinée à un smartphone Android, soit du J2SE. Les bundles composant le bootstrap sont eux par contre génériques.

Le bootstrap est composé des bundles Main, RabbitMQ Client, BasicProbes (cf. figure 6.1) qui sont propres au système de déploiement. Le bundle Main est le point d'entrée du bootstrap. Le bundle RabbitMQ Client permet la liaison avec le domaine de déploiement, concrètement avec un serveur RabbitMQ [Rab] qui gère le domaine (RabbitMQ est un middleware de communication par messages, cf. 8.2.2). Le bundle BasicProbes contient un ensemble de sondes basiques (cf. section suivante 8.2.1.1) ainsi présentes sur chaque appareil.

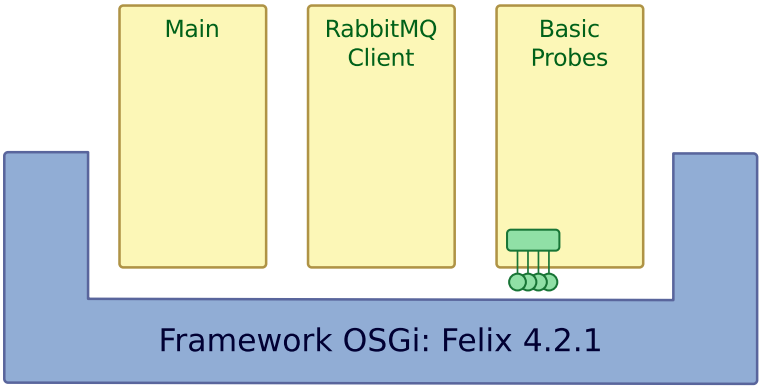


Figure 8.2 – Architecture du bootstrap

EX24

En installant le bootstrap, l'utilisateur qui est aussi l'administrateur du système donne les droits nécessaires au support local de déploiement . Le choix d'un framework OSGi (et par extension, de la plateforme Felix) permet de restreindre la phase de configuration du bootstrap demandée à l'utilisateur au seul choix entre deux programmes, en fonction du type d'appareil visé.

À titre d'illustration, les deux versions du bootstrap utilisables dans le projet INCOME sont disponibles à l'adresse suivante : `http://www.irit.fr/income/index.php?page=tache-5`. Pour les applications Android, le bootstrap est mis à disposition de l'utilisateur par l'intermédiaire d'un QR code, contenant l'adresse de téléchargement, lui évitant d'entrer manuellement cette adresse.

8

8.2.1.1 Sondes

Sondes basiques Le bootstrap contient un bundle de sondes basiques. Ces sondes sont celles qui sont les plus susceptibles d'être utilisées par le concepteur, permettant de récupérer des informations logicielles et matérielles de l'appareil.

Les sondes basiques, implémentées avec l'aide d'étudiants de Master (projet de Master 1 et stage de Master 2), sont :

— CPU : mesure de la fréquence du processeur, etc.
— RAM : mesure de la mémoire vive libre, de mémoire vive totale, etc.
— HD : mesure de l'espace disque libre, espace disque total, etc.
— OS : reconnaissance du système d'exploitation, de la version, etc.
— Network : reconnaissance de l'adresse IP, du type de connexion, etc.
— Locale : identification des paramètres régionaux.
— Peripheral : liste des périphériques
— JVMMemory : mesure de la mémoire de la JVM (*Java Virtual Machine*, machine virtuelle Java) libre, totale, etc.
— Screen : liste des écrans, taille de l'écran, etc.
— Java : nom, version, vendeur, etc.

Le paquet fr.enac.lii.basicprobes est une implémentation du bundle OSGi décrit ci-dessus et schématisé dans la figure 6.2. Il permet d'offrir en REST [RES] (sur l'hôte local, sur le port 8080 et à l'adresse */status/*, `localhost:8080/status`) l'ensemble des informations retournées par les sondes, dans un format JSON [JSO].

Pour cela, il est nécessaire de disposer dans l'environnement OSGi des bundles qui apparaissent dans la figure 8.3. Jetty [Jet] est un conteneur de *servlets* (implémentation par défaut de conteneur de *servlets* dans Felix). L'enregistrement des *servlets* est simplifié en utilisant le service *whiteboard*, d'où le bundle correspondant. Pour une exécution correcte, Jetty a besoin d'un bundle gérant les logs : pour cela, nous avons choisi l'implémentation par défaut (Felix.log). La sérialisation des données au format JSON est réalisée par le framework Google Gson [Gso] directement disponible sous forme de bundle OSGi.

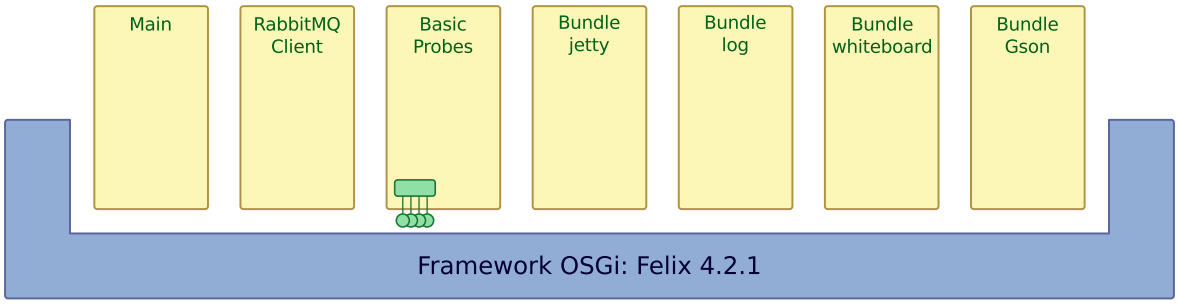


Figure 8.3 – Architecture complète du bootstrap

Framework des sondes Les sondes dépendent du paquet fr.irit.devext.probesframework qui permet d'abstraire les interactions avec des sondes. Il a été produit à partir de celui fourni par des stagiaires Master 1 de l'Université Paul Sabatier en 2013.

Le framework des sondes consiste principalement en une classe abstraite qui doit être spécialisée pour chacune des sondes. Il permet ensuite la vérification de l'existence et de

l'activité d'un appareil, ainsi que la récupération d'informations sur cet appareil. De plus, il est possible de définir des méthodes de rappel (*callback*) paramétrées qui permettent à la sonde de signaler une variation significative. Cette fonctionnalité est discutée plus en détails à la section 8.2.8. Le diagramme de classes en UML est présenté dans la figure 8.4 ainsi qu'un diagramme de séquence dans la figure 8.5.

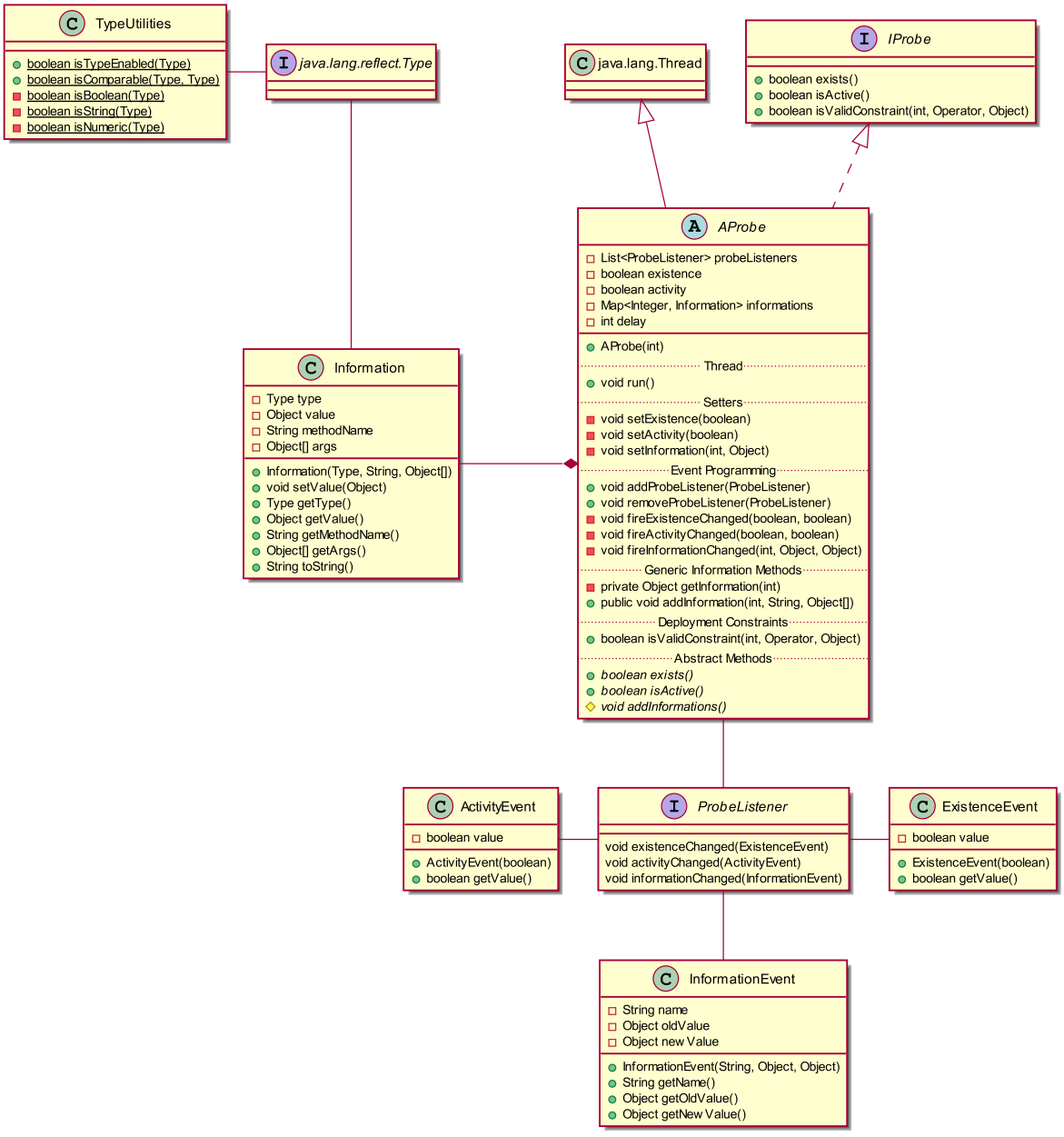


Figure 8.4 – Diagramme de classes UML du framework de sondes

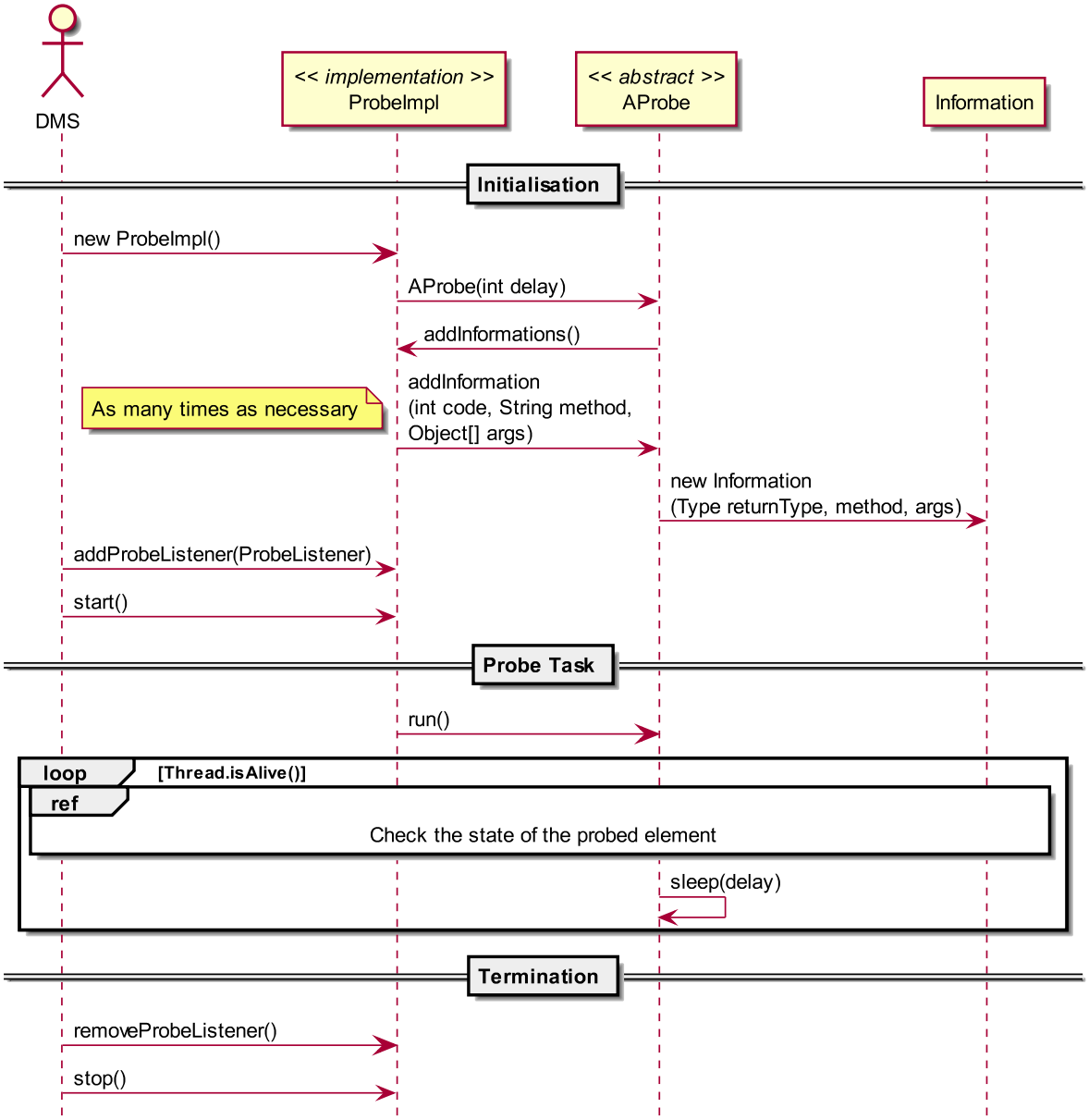


Figure 8.5 – Diagramme de séquences UML du framework de sondes

Un exemple trivial de sonde fournissant les paramètres régionaux est donné dans le listing 8.1.

```java
package fr.enac.lii.basicprobes.probes;

import java.util.Locale;
import fr.irit.devext.probesframework.AbstractProbe;
import fr.irit.devext.probesframework.IncorrectTypeException;

public class LocaleProbe extends AbstractProbe {

    private Locale locale = Locale.getDefault();

    public enum Information {
        LOCALE_LANGUAGE, LOCALE_COUNTRY
    }

    public LocaleProbe() {
        super(1000 * 60 * 60);
    }

    @Override
    protected void addInformation() {
        try {
            addInformation(Information.LOCALE_LANGUAGE.ordinal(), "language");
            addInformation(Information.LOCALE_COUNTRY.ordinal(), "country");
        } catch (NoSuchMethodException e) {
            errorLog(e.toString());
        } catch (IncorrectTypeException e) {
            errorLog(e.toString());
        }
    }

    @Override
    public boolean exists() {
        return true;
    }

    @Override
    public boolean isActive() {
        return true;
    }

    /**
     * Get the language of the device
     *
     * @return String the language
     */
    public String language() {
        return locale.getDisplayLanguage();
    }

    /**
     * Get the country of the device
     *
     * @return String the country
     */
    public String country() {
        return locale.getDisplayCountry();
    }
}
```

Listing 8.1 – Exemple de code de sonde des paramètres régionaux

8 EX17

Sondes définies par le concepteur Le framework des sondes permet aussi de définir des sondes autres que les sondes basiques. Si le concepteur a besoin d'informations supplémentaires concernant l'appareil, il peut définir ses propres sondes. Pour cela, il lui suffit de créer une nouvelle sonde, au même format que la sonde présentée dans le lis-

ting 8.1 (c'est-à-dire qui doit contenir l'énumération `Informations`, etc.). La nouvelle sonde doit aussi contenir les méthodes propres qui seront celles utilisées dans le descripteur pour la définition des conditions des critères basiques. Le langage Java n'est pas imposé pour l'écriture des sondes. Seule la classe principale doit être en Java, celle qui utilise le framework des sondes. Le concepteur peut faire en interne dans la sonde un lien avec un autre langage et utiliser ce lien dans la classe principale.

Les sondes multi-échelles diffèrent des sondes normales de part les informations qu'elles fournissent. Une sonde multi-échelle est définie par point de vue. La sonde contient les méthodes pour accéder aux dimensions, aux échelles et aux instances d'échelle. Ainsi il est possible de récupérer plusieurs informations multi-échelles concernant un appareil : l'appartenance à une échelle d'un appareil ou dans quelle instance d'échelle il se trouve.

8.2.1.2 Android et J2SE

Les deux versions du bootstrap sont une application native Android et une application Java standard (J2SE), contenant les éléments décrits dans la figure 6.4. Elles sont distribuées sous leurs formats natifs (.apk et .jar). Elles contiennent un moyen de démarrer l'implémentation d'OSGi choisie (Felix), et d'en contrôler l'exécution. C'est la seule partie de code du système de déploiement à être spécifique à une plateforme.

Dans les deux cas, le bootstrap utilise le système d'exploitation de l'appareil pour indiquer à l'utilisateur son état. On retrouve ainsi à l'exécution, dans la barre des tâches (ou l'équivalent sous Android), une icône permettant de vérifier l'état du système de déploiement, de l'arrêter, etc.

La figure 8.6 montre une image de l'application Android : à gauche on voit le logo INCOME dans la barre des tâches. En cliquant sur le logo, on accède à une vue de l'application qui montre son état (UUID, nombre de bundles OSGi actifs et leur état, la valeur 32 indiquant que le bundle est actif), et qui permet d'arrêter le service.

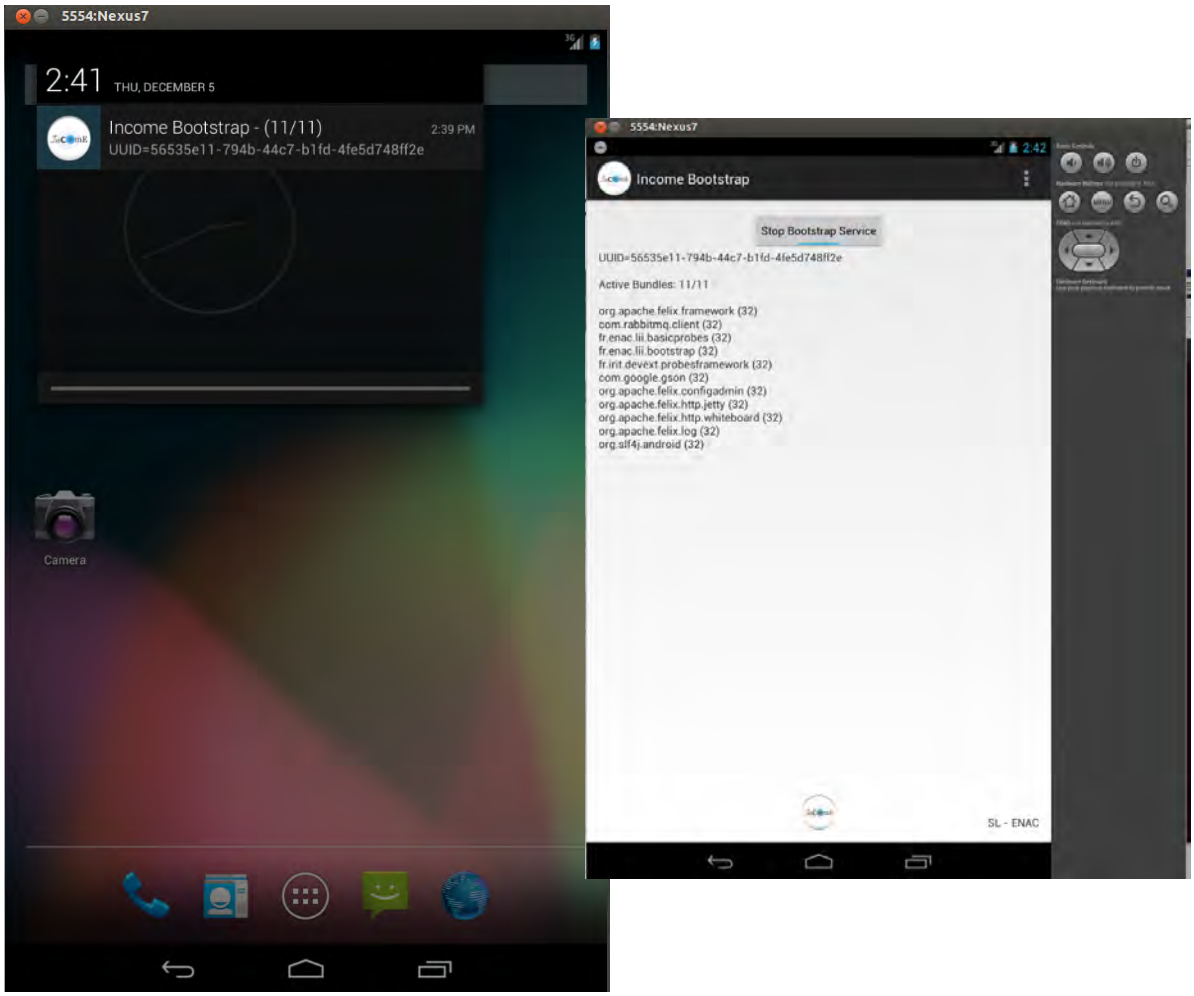


Figure 8.6 – Application de bootstrap pour Android

8.2.2 Inscription et enregistrement des appareils

Le gestionnaire de domaine de déploiement est dans notre cas un ensemble d'instances de serveurs RabbitMQ [Rab] (protocole AMQP [AMQ]) et un serveur agrégeant la liste des appareils disponibles. RabbitMQ permet de construire un protocole de présence et supporte la communication en mode asynchrone entre les appareils répartis à toutes les échelles du réseau (passage de firewalls/routeurs, etc.). Ainsi, il est possible de demander à distance l'installation, l'activation ou l'arrêt d'un bundle, ou encore l'état de l'appareil en invoquant le service de sondage basique.

EX16 EX17

Pour détecter la disparition, nous utilisons un système de signalement *heartbeat* (battement de cœur). C'est un signal périodique qui permet de s'assurer de la présence des appareils dans le domaine de déploiement.

Nous avons réimplémenté un service de présence simple, dans lequel chaque client, identifié par un identifiant unique (UUID), dépose périodiquement cet identifiant dans la file de présence.

De ce fait, il faut également disposer d'un serveur (dont une instance tourne sur le serveur « flabelline.com » pour l'instant). Ultérieurement, il sera possible de faire tourner Rab-

bitMQ sur des appareils sur le cloud. RabbitMQ est extensible par nature car il est possible de constituer un cluster de serveurs RabbitMQ. La librairie cliente assez légère (350 Ko) est livrée directement sous la forme d'un bundle OSGi.

La figure 8.7 présente les échanges entre le serveur RabbitMQ et le bundle Main du bootstrap. Lors de l'enregistrement, un thread de présence signale au serveur RabbitMQ la présence de l'appareil sur le domaine. Sur le serveur RabbitMQ, les évènements ont une durée de vie (TTL, *Time To Live*) de 60 secondes. Lorsque le moniteur de déploiement envoie une commande à un appareil, le serveur RabbitMQ envoie une commande *DMS Order* préfixée par l'UUID de l'appareil. Le bootstrap traite cette commande et renvoie une réponse, en préfixant toujours son message par son UUID, pour garder la traçabilité du traitement des commandes. Le *DMSShell* est un interpréteur de commandes qui permet le traitement des messages envoyés entre le bootstrap et le moniteur.

EX10 EX11

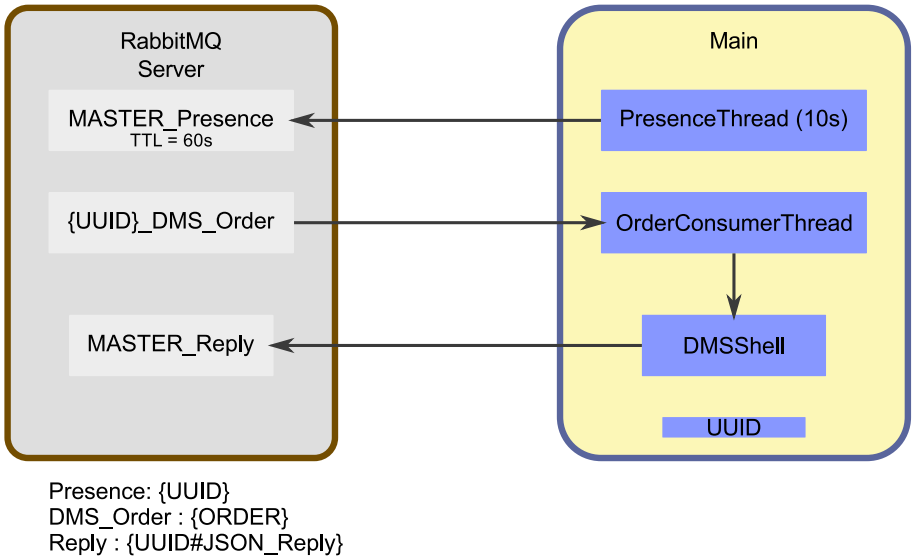


Figure 8.7 – Interactions entre le bundle Main et serveur RabbitMQ

8.2.3 Expression des propriétés de déploiement

EX3 EX7 EX19 EX31 EX33

Le DSL MuScADeL permet la spécification du déploiement des systèmes répartis multi-échelles. MuScADeL permet de décrire les types de composants à déployer, ainsi que leurs contraintes propres, les informations concernant les sondes utilisées pour la récupération de l'état du domaine, et les exigences de déploiement.

Un environnement de développement intégré de MuScADeL facilite l'expression du déploiement pour le concepteur. Un plugin Eclipse pour MuScADeL permet d'utiliser cet environnement. Le plugin a été conçu avec le framework Xtext [Xte2]. Xtext permet, avec la spécification de la grammaire du langage d'obtenir un plugin avec coloration syntaxique, ainsi que des outils de validation du langage, comme la détection d'éléments non déclarés, l'auto-complétion sensible au contexte, etc . Le framework Xtext permet aussi la gestion de la séparation du code en plusieurs fichiers et l'inclusion de fichiers. Les fichiers MuScADeL ont pour extension *.musc*. Le plugin MuScADeL dans Eclipse est présenté dans la figure 8.8. Nous pouvons y voir les différents fichiers composant l'exemple présenté dans le chapitre 5.

EX32

Le framework Xtext fournit aussi des outils pour personnaliser l'auto-complétion et

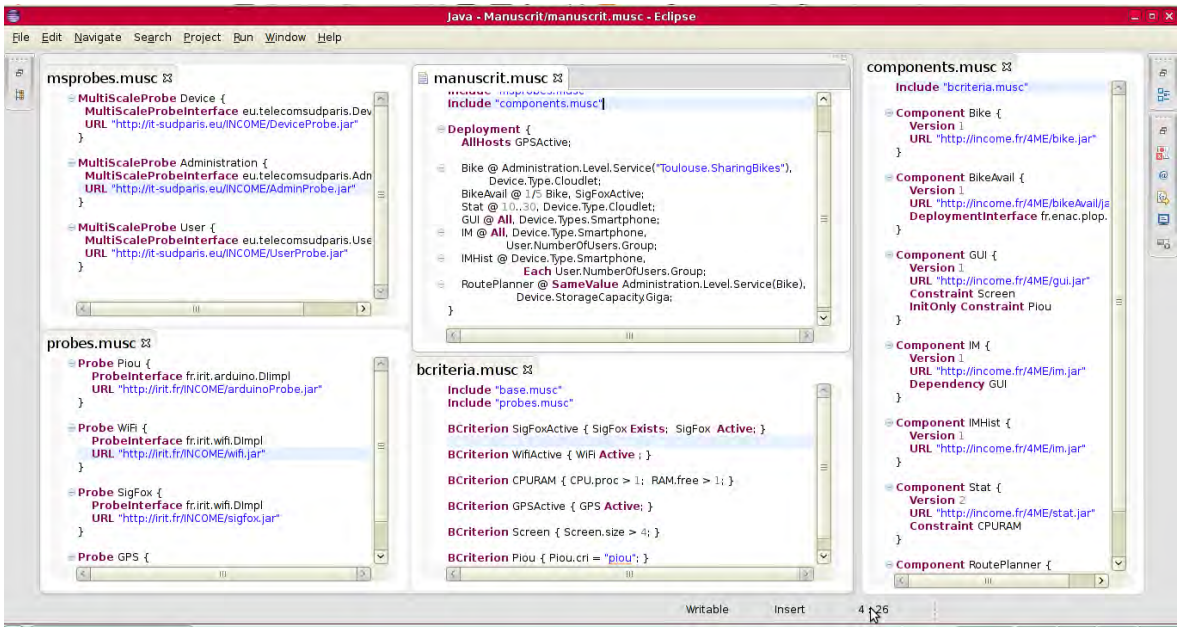


Figure 8.8 – Éditeur MuScADeL dans Eclipse

la validation des éléments du langage. Nous l'utilisons pour les critères multi-échelles. Lorsque le concepteur ajoute dans le projet un fichier de modèles spécifique MuSCa (un fichier *.mscharacterization*), le plugin utilise ce fichier pour proposer une complétion des dimensions et des échelles, ainsi que pour valider leur utilisation (cohérence entre le point de vue, la dimension et l'échelle). La figure 8.9 montre cette complétion d'échelle.

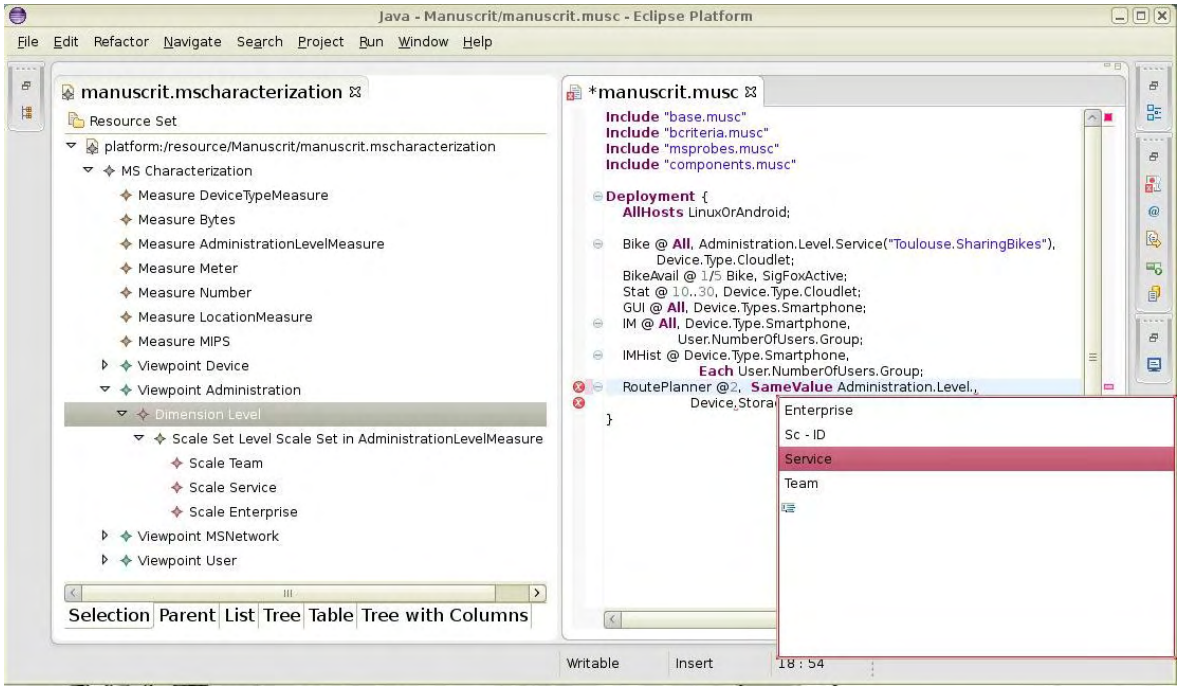


Figure 8.9 – Utilisation de la caractérisation MuSCa dans le plugin de MuScADeL

8.2.4 Analyse du descripteur MuScADeL

Le framework Xtend [Xte1] permet la génération de code à partir de l'arbre syntaxique du langage défini. En parcourant l'arbre syntaxique de MuScADeL, nous générons du code Java qui sert à la récupération de l'état du domaine, à la construction des contraintes à donner en entrée au solveur de contraintes, et au lancement du déploiement. Pour cela nous utilisons une bibliothèque que nous avons définie, `MuScADeLDeploymentPlan`. Cette bibliothèque contient le code du DMSMaster, l'interface qui permet d'interagir avec le système de déploiement pour le déploiement initial. En fonction du descripteur MuScADeL, le code généré utilise cette bibliothèque pour préciser quelles sont les sondes utiles et quelles sont les propriétés définies afin d'ajouter les contraintes spécifiques. Tous les éléments du langage sont représentés dans cette bibliothèque.

Le code Java correspondant au descripteur MuScADeL est généré à chaque fois qu'un fichier MuScADeL est enregistré. Il est généré dans un projet existant appelé GenerateDeploymentPlan. Les noms des projets et fichiers MuScADeL sont réutilisés afin que le concepteur puisse retrouver le code correspondant à son projet courant.

8.2.5 Récupération de l'état du domaine

L'étape de récupération de l'état du domaine début par la récupération du descripteur des sondes utilisées et des critères et conditions à satisfaire. Ces deux éléments sont construits avec la génération du code Java, et donnés en entrée à la bibliothèque `MuScADeLDeploymentPlan`. Si des sondes spécifiques sont fournies par l'utilisateur, en plus des sondes basiques, la bibliothèque envoie une commande d'installation de ces sondes spécifiques au support local du déploiement sur les appareils. Une commande est aussi construite et envoyée afin de demander les informations de l'appareil. Cette commande contient le nom de la sonde et de la méthode à appeler : nom de la méthode pour les sondes normales, dimension ou dimension/échelle pour les sondes multi-échelles. Chaque appareil envoie sa réponse. Puis, en fonction des valeurs contenues dans les conditions, une vérification de satisfaisabilité s'effectue sur le moniteur de déploiement. Cette étape de vérification permet la construction de la matrice *Dom*, qui est l'une des entrées de la phase de résolution de contrainte (cf. section 6.3.3). Les commandes sont envoyées en utilisant les serveurs RabbitMQ.

EX13 EX20 EX35

8.2.6 Résolution des contraintes

La matrice *Dom* calculée à partir des résultats des sondages, et la matrice *Comp* à partir des données construites par le code généré, la phase de résolution peut être lancée. Pour cela, nous utilisons une bibliothèque que nous avons développée, `MuScADeLSolving`. Cette bibliothèque offre des méthodes pour l'ajout de propriétés de déploiement, qu'elle traduit dans des appels de méthodes. Ces appels de méthodes construisent le problème à résoudre pour un solveur de contraintes.

8.2.6.1 Solveur de contraintes

Afin de pouvoir choisir un solveur de contraintes qui supporte la phase de résolution, nous avons étudié et comparé un certain nombre d'outils candidats : Cream [Tam2], Co-

pris [Tam1], JaCoP [KS], or-tools [ort], jOpt [jOp], et Choco [C.H.O.C.O. Team, 2010]. Nous avons considéré le type de problème traité et le support disponible en matière d'évolution et de documentation. La table 8.1 présente les résultats de cette comparaison. Tous les solveurs sont compatibles avec Java (ils sont soit écrits en Java, soit interfaçables avec Java). Les acronymes CSP, COP, CP et JS correspondent respectivement à *Constraint Satisfaction Problem*, *Constraint Optimization Problem*, *Constraint Problem* et *Job Scheduling*.

	Problème	Maintenance	Documentation
Choco	CSP	maintenu	complète
Copris	COP, CSP, CP	maintenu	quasi inexistante
Cream	CSP	obsolète (2008)	légère
JaCoP	CSP	maintenu	existante
jOpt	CSP, JS	maintenu	inexistante
or-tools	CSP	maintenu	quasi inexistante

Tableau 8.1 – Comparaison des solveurs de contraintes

Notre problème étant un problème de satisfaction de contrainte (CSP), la classe de problème n'a pas été le critère de choix. Parmi ces solveurs, nous avons choisi **Choco** [C.H.O.C.O. Team, 2010]. Nous avons déjà une certaine expertise sur cet outil, et la librairie est complète et simple à utiliser.

8.2.6.2 `MuScADeLSolving`

Dans cette section, nous présentons la librairie de résolution de contraintes issues des propriétés de déploiement `MuScADeLSolving`. L'interface `MuscadelSolvingInter` de la librairie (listing 8.2) présente les méthodes accessibles pour l'ajout des contraintes : `cardinaliteSimple`, `cardinaliteIntervalle`, `cardinaliteAll`, `ratio`, `sameValue`, `differentValue`, `each` et `solving`.

```
1  public interface MuscadelSolvingInter {
2      public void cardinaliteSimple (int cmp, int card);
3      public void cardinaliteAll (int cmp);
4      public void cardinaliteIntervalle(int cmp, int min, int max);
5      public void ratio(int cmpP, int cmpS, int ratioP, int ratioS);
6      public void sameValue(int cmp1, int cmp2, String[] sonde);
7      public void differentValue(int cmp1, int cmp2, String[] sonde);
8      public void each (int cmp, String[] sonde);
9      public int[][] solving() throws MuscadelSolvingExc;
10 }
```

Listing 8.2 – Interface `MuscadelSolvingInter`

La classe `MuscadelSolving` contient les matrices *Comp* et *Dom*, ainsi que le modèle Choco et la matrice de possibilités *SatVar*. Cette dernière est construite lors de la construction d'un objet `MuscadelSolving`, avec un appel à une méthode privée dans le constructeur.

La méthode `cardinaliteSimple` ajoute une contrainte de cardinalité simple, telle que décrit par la formule (6.2).

La méthode `cardinaliteIntervalle` ajoute les contraintes de cardinalité à intervalle (par exemple, dans le listing 5.5, à la ligne 7), tel que décrit par la formule (6.3).

La méthode `cardinaliteAll` ajoute les contraintes de cardinalité `All`, tel que décrit par la formule (6.4). Comme pour la méthode `cardinaliteIntervalle`, la méthode `cardinaliteAll` n'ajoute pas que des contraintes. Une contrainte est ajoutée pour spécifier qu'au moins un composant doit être déployé dans le domaine, et la ligne du composant dans la matrice `satVar` est ajoutée à la liste des lignes à maximiser.

La méthode `ratio` ajoute une contrainte de ratio entre composants, tel que décrit par la formule (6.5). Elle prend en paramètre les deux composants sur lesquels se portent le ratio, le numérateur et le dénominateur.

Les méthodes `samevalue` et `differentValue` ajoutent les contraintes relatives à la dépendance entre composants, tel que décrit dans la formule (6.7). Elles prennent en paramètres les composants concernés et le tableau des données du sondage (par exemple, les tableau 6.2a et 6.2b).

La méthode `each` ajoute les contraintes de placement d'un composant par instance d'échelle, tel que décrit par la formule (6.8). Elle prend en paramètre le composant concerné et un tableau des données de sondage.

La méthode `solving` lance la résolution du solveur de contrainte. Si aucune ligne de la matrice `satVar` ne doit être maximisée, la résolution est directement lancée. Sinon, les directives de maximisation sont ajoutée puis la résolution est lancée. Par la suite, la faisabilité du problème est vérifiée : si le problème n'a pas de solution, une exception `MuscadelSolvingExc` est levée, sinon, la première solution est récupérée, et est retournée par la méthode.

La librairie est décrite plus en détails dans [Boujbel et al., 2014a].

8.2.6.3 Génération du plan de déploiement

La construction des éléments du langage en utilisant la bibliothèque `MuScADeLDeploymentPlan` permet l'ajout transparent de propriétés de déploiement au problème Choco[1] en utilisant la bibliothèque `MuScADeLSolving`. En effet, chaque classe correspondant à une propriété contient une méthode qui ajoute la propriété représentée à l'instance de `MuscadelSolving`. Une simple boucle sur l'ensemble des instances correspondant aux propriétés permet l'ajout de l'ensemble des contraintes au problème du solveur de contraintes. Puis, la résolution est lancée. À la fin de cette étape, un plan de déploiement, sous la forme d'une matrice d'obligations $Composant * Appareil$ est disponible, et est présenté à l'utilisateur dans une fenêtre.

EX13 EX34 EX35 EX36

8.2.7 Installation et activation

Une fois le plan de déploiement obtenu, l'opérateur peut lancer le déploiement du système. En utilisant les serveurs RabbitMQ, une commande est envoyée aux différents appareils, contenant les différents composants à installer et à activer.

EX5 EX12 EX13 EX37 EX42

1. Le problème Choco est un ensemble de variables auxquelles des contraintes sont ajoutées. Dans notre cas, il s'agit des variables contenues dans la matrice *SatVar* définie dans la section 6.3.2.1.

8.2.8 Déploiement dynamique

EX8 EX15 EX16 EX21 EX38

Le système de déploiement surveille le domaine et l'état du domaine à l'exécution de l'application. Si un nouvel appareil entre dans le domaine de déploiement, il s'enregistre sur le serveur RabbitMQ. Le moniteur de déploiement s'aperçoit qu'un nouvel appareil est dans le domaine de déploiement, il récupère ses caractéristiques en y installant les nouvelles sondes si nécessaires. Puis, il vérifie dans la liste des composants ayant une propriété dynamique si l'appareil satisfait toutes les propriétés du composant. Si oui, il déploie dessus le composant concerné. Pour le moment cette étape se fait de manière centralisée. À terme, cette vérification de satisfaction de propriétés se fera localement avec le support local du système de déploiement, et ce sera le superviseur hiérarchique qui piloterait l'installation et l'activation.

EX22 EX23 EX39

La surveillance de l'état du domaine permet de détecter les pannes provenant de la variation de l'état des appareils. Cette supervision peut se faire localement, en utilisant le framework de sondes présenté dans les figures 8.4 et 8.5. Le support local du système de déploiement réalise cette supervision, et disposant des valeurs seuils, il peut détecter la violation de propriété et en référer à son superviseur hiérarchique.

8.2.9 Bilan

Dans cette section, nous avons exposé nos différentes réalisations et les avons mises en relation avec les exigences du déploiement de systèmes répartis multi-échelles. Comme nous pouvons le constater, nous avons 45 exigences de départ. Dans cette section, nous montrons comment nous répondons à 37 de ces exigences. Les exigences restantes, comme **EX25**, **EX26**, **EX27**, relèvent essentiellement de l'ingénierie. En effet, pour ces exigences, il faut développer les composants du système de déploiement qui s'interfacent avec les machines passerelles de chaque infrastructures ciblées.

8.3 Démonstrations

Dans cette section, nous présentons des démonstrations d'utilisation des différents éléments de notre prototype.

8.3.1 Récupération de l'état d'un domaine de déploiement

En juin 2014, nous avons présenté une démonstration nommée « Récupération de l'état d'un domaine de déploiement » [Kem et al., 2014] lors de la conférence francophone UbiMob. La démonstration présente le bootstrap et l'étape de récupération de l'état du domaine de déploiement. Le bootstrap est lancé sur différentes plateformes (Windows, machines virtuelles Ubuntu et Mac OS, émulateur Android), et le moniteur de déploiement sur Windows. Les bootstraps s'enregistrent un à un auprès du gestionnaire de déploiement et sont visibles sur l'interface graphique du moniteur de déploiement. Puis, la commande de récupération de l'état du domaine général, c'est-à-dire récupération des informations des sondes basiques, est envoyée. Le moniteur affiche donc une à une les réponses qu'il reçoit des différents appareils. La figure 8.10 montre cette dernière étape.

Cette vidéo présente aussi la plateforme de test que nous avons mise en place, constituée de quatre machines virtuelles sous :

— Windows : XP (32bits), 7 (64bits)
— Linux : Ubuntu 12.04 (32bits)
— Mac : Mac OS X 10.8 Mountain Lion (64bits)
— Android : SDK Version 16

Il s'agit de machines virtuelles utilisant le logiciel VMware [VMware Inc., 2008], à part pour Android, où nous utilisons l'émulateur fournit par le SDK.

Cette démonstration est accessible, sous la forme d'une vidéo, à l'adresse `http://ubimob2014.sciencesconf.org/39370`.

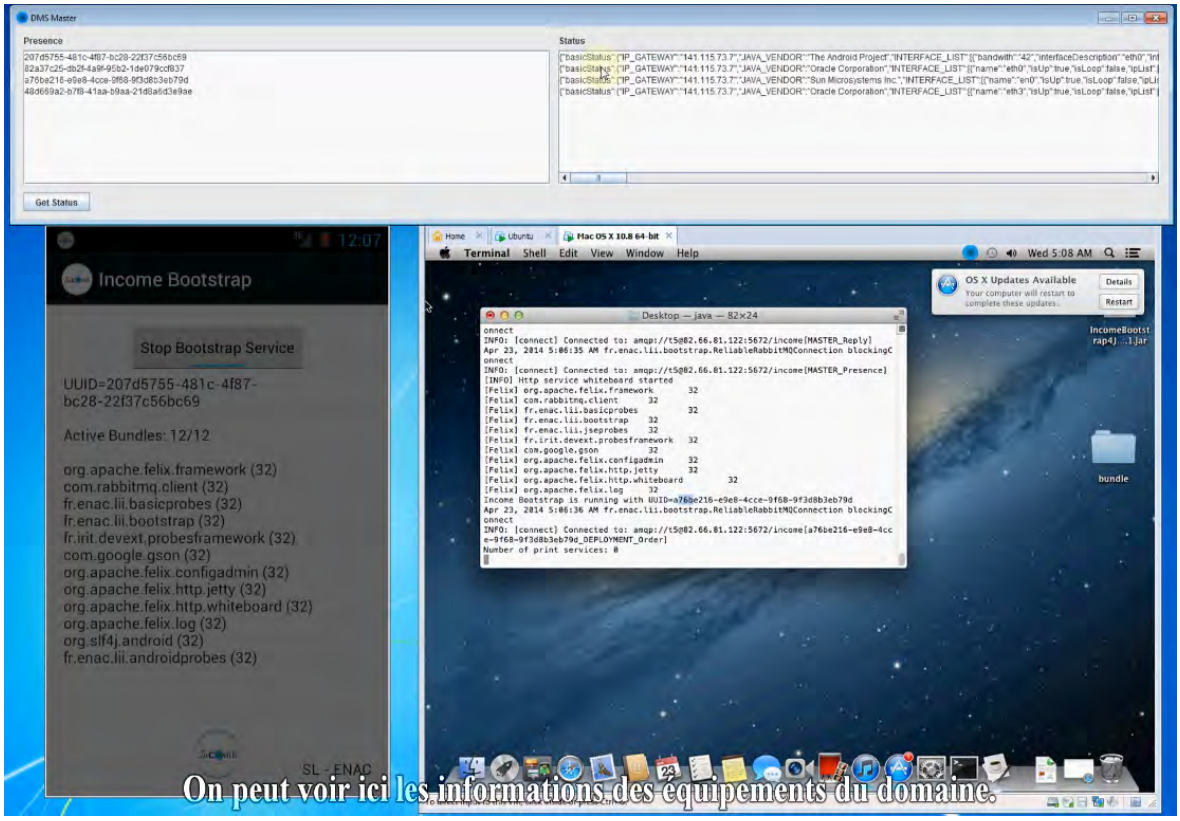


Figure 8.10 – Extrait de la démonstration présentée à UbiMob'14 [Kem et al., 2014]

8.3.2 Génération d'un plan de déploiement

Lors des réunions plénières du projet INCOME, nous faisons état de l'avancement de nos travaux. Afin d'illustrer ces présentations, nous avons effectué des démonstrations et réalisé des vidéos mettant en scène les différentes étapes d'évolution du prototype.

Une de ces vidéos présente le plugin MuScADeL et la génération d'un plan de déploiement. Elle commence par montrer la grammaire de MuScADeL dans Xtext, puis le lancement d'un Eclipse avec le plugin de MuScADeL. Un projet MuScADeL est alors présenté. Il est composé de différents fichiers, dont le fichier décrivant les sondes basiques. Les différents éléments du langage sont présentés ainsi que la validation, qui est visible par l'utilisation d'un sonde définie (sonde `Locale`). Nous pouvons aussi voir que l'ajout d'un

fichier MuSCa de caractérisation multi-échelle permet d'avoir la complétion automatique des dimensions et des échelles du point de vue Device. Lors de l'enregistrement des fichiers *.musc*, le code Java correspondant est automatiquement généré. Ce code Java lance le moniteur de déploiement, et donc la génération du plan. Les différents appareils se connectent au moniteur de déploiement et la génération du plan de déploiement est lancée. Le plan de déploiement généré en fonction de l'état du domaine est visible dans une fenêtre séparée. La figure 8.11 montre le moniteur de déploiement et le plan généré.

La vidéo est disponible à l'adresse suivante `http://anr-income.fr/T5/MuScADeL_IDE_Deployment_Plan_Generation.mkv`.

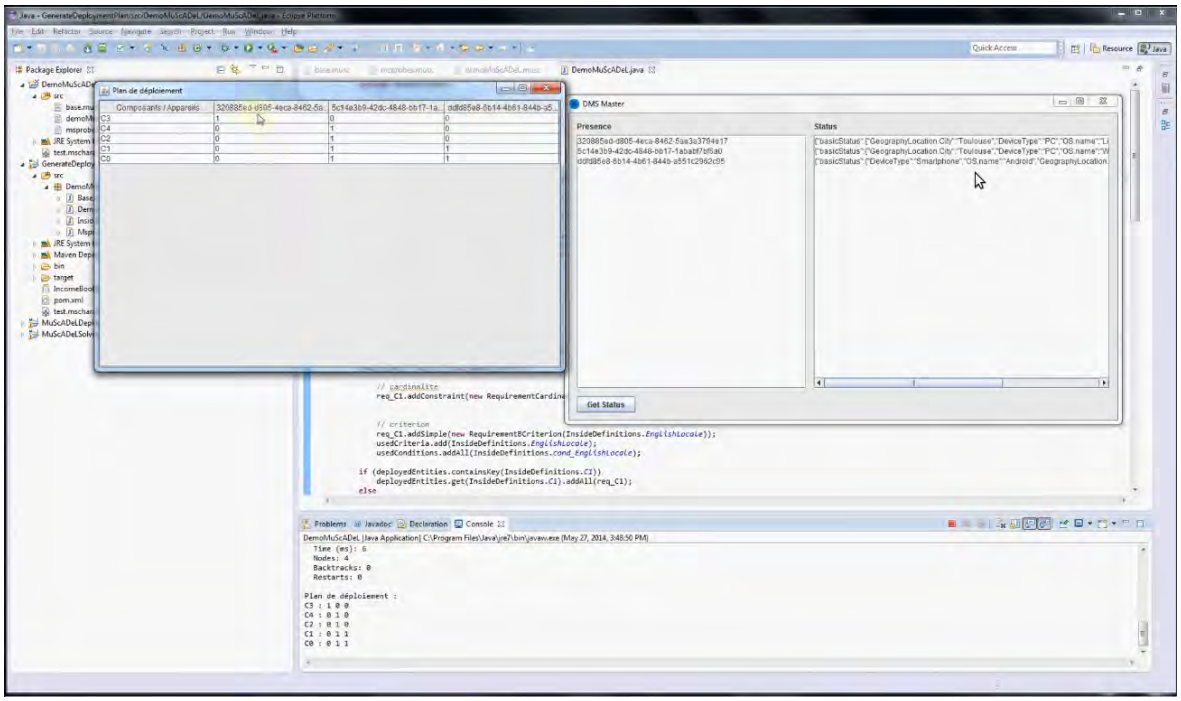


Figure 8.11 – Extrait de la vidéo de génération d'un plan de déploiement

8.3.3 Tutoriel pour la prise en main de MuScADeL et MuScADeM

Un tutoriel facilite la prise en main de MuScADeL et MuScADeM. Il est disponible à l'adresse suivante : `http://www.irit.fr/~Raja.Boujbel/muscadel/tutoriel.html`. Le tutoriel assiste l'utilisateur dans l'installation du plugin dans Eclipse, fournit les différents bootstraps, et explique la prise en main de MuScADeM. Une fois le plugin installé, l'utilisateur peut écrire son premier descripteur MuScADeL puis lancer le déploiement de composants fournis (simples *Hello World*, une version J2SE et une version Android). Le tutoriel décrit aussi comment développer ses propres sondes, en utilisant le framework de sondes. Toutes les dépendances des projets (`MuScADeLDeploymentPlan`, `MuScADeLSolving`, framework des sondes, etc.) sont disponibles sur un dépôt Maven, ce qui facilite la configuration du plugin et du framework. Une dizaine de personnes ont utilisé MuScADeL et MuScADeM lors de la dernière plénière du projet INCOME en septembre 2014.

8.4 Performances et passage à l'échelle

8.4.1 Génération du plan de déploiement

Nous avons effectué quelques tests de performances de la génération du plan de déploiement. Pour cela, nous avons défini un ensemble de méthodes permettant de générer des matrices *Dom* et *Comp* de différentes tailles, afin d'analyser l'évolution du temps de résolution en fonction du nombre de composants et d'appareils. Les tests ont été effectués sur un ordinateur ayant processeur Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz.

La figure 8.12 montre l'évolution du temps de résolution en fonction du nombre d'appareils et du nombre de composants. Dans tous les cas, le nombre de composants et d'appareils croit à chaque itération. Les matrices *Comp* et *Dom* ont tous leurs coefficients à 1. Les courbes représentent différents cas :

- simple : l'exigence de cardinalité simple, pour laquelle la cardinalité demandée est la moitié du nombre d'appareils ;
- intervalle : l'exigence d'intervalle, pour laquelle l'intervalle demandé est le quart du nombre d'appareils ;
- ratio : l'exigence de ratio, pour laquelle le ratio demandé est de un pour quatre, pour la moitié des composants déjà déployés ;
- each : l'exigence `Each`, pour laquelle le nombre d'instances croit en fonction du nombre d'appareils ;
- same_device : la contrainte de localité d'un composant par rapport à un autre.

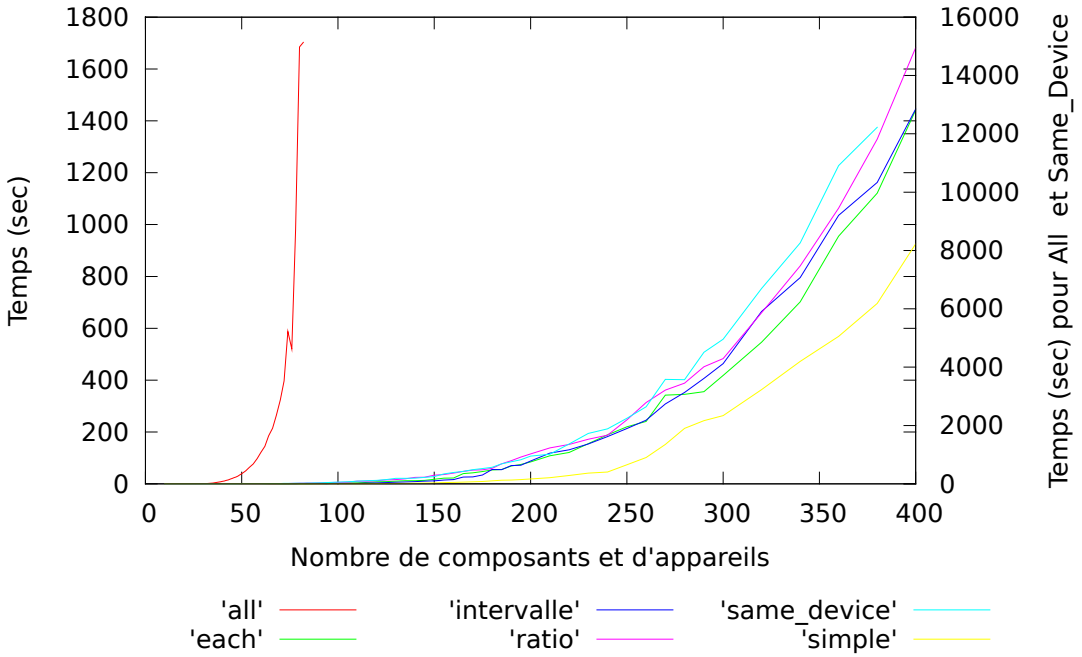


Figure 8.12 – Temps de calcul du plan de déploiement en variant le nombre de composants et d'appareils

La propriété qui demande le plus grand temps de calcul est propriété `All`. Cela s'explique par la manière dont cette propriété est gérée pas Choco. Aucune contrainte n'est exprimée. Par défaut, le solveur donne la solution la plus simple, qui est de ne pas déployer ce

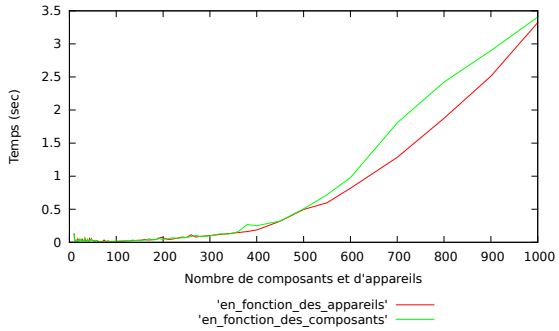
composant. Or nous voulons que le composant soit déployé sur le maximum d'appareils qui satisfasse ses propriétés. Nous demandons donc au solveur de maximiser le nombre de composant à déployer. La maximisation d'une variable est très coûteuse. En effet, afin de maximiser une variable, le solveur cherche si une solution existe avec le plus petit nombre possible, si oui, il augmente son objectif, si non retourne le précédent objectif. Ainsi il cherche la faisabilité et une solution à chaque pas d'une variable à optimiser. L'augmentation du temps est donc exponentielle. Pour une matrice de 5 composants et appareils, le solveur recherche 25 solutions avant de trouver celle qui maximise les variables, et pour une matrice de 200 composants et appareils, il en recherche 40000.

La contrainte de localité au même appareil demande aussi beaucoup de temps de calcul. Cela vient de la manière dont cette contrainte est exprimée. L'expression donnée à Choco est une disjonction entre les coefficients de deux lignes de la matrice des variables. Nous avons essayé d'optimiser cette contrainte en appliquant la loi de De Morgan, mais cela n'améliore pas les performances.

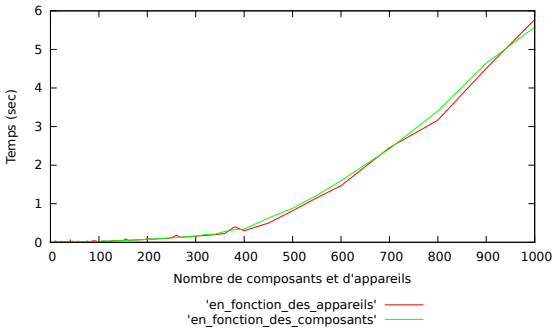
Dans un deuxième temps, nous avons étudié l'impact du nombre de composants et du nombre d'appareil sur le temps de calcul. Pour cela, dans un cas nous fixons le nombre de composants à 10, et dans une autre le nombre d'appareils à 10. Pour ces tests aussi nous avons défini les matrices *Dom* et *Comp* avec tous les coefficients à 1. Le figure 8.13 montre l'évolution du temps de calcul en fonction de l'augmentation du nombre de composant ou du nombre d'appareils. Les figures représentent :

- la figure 8.13a : l'exigence de cardinalité simple, avec comme cardinalité la moitié des appareils du domaine ;
- la figure 8.13b : l'exigence d'intervalle, avec un intervalle s'étendant à la moitié des appareils ;
- la figure 8.13c : l'exigence de ratio, avec un ratio sur la moitié des composants, avec une expression de ratio d'un quart ;
- la figure 8.13d : l'exigence `All`, sur l'ensemble des composants.
- la figure 8.13e : l'exigence `Each`, avec une nombre d'instance d'échelle correspondant au quart du nombre d'appareil ;
- la figure 8.13f : la contrainte de localité (deux composants sur le même appareil) ;

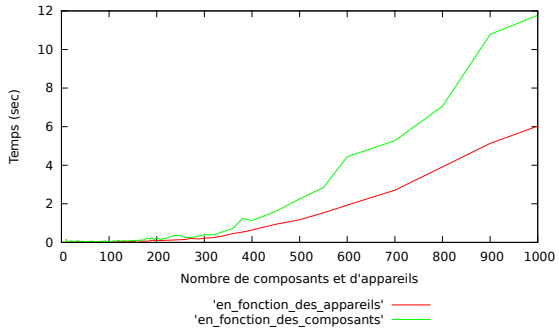
Nous pouvons remarquer que globalement, le fait de fixer un des paramètres, réduit considérablement le temps de calcul. Pour la plupart des propriétés, la différence de temps de calcul n'est pas notable. En revanche, pour la propriété `All`, un gain de temps est visible si le nombre d'appareils est fixe. Pour la propriété de ratio, c'est l'inverse, le calcul est plus rapide si seul le nombre d'appareils croît (nombre de composants fixe).



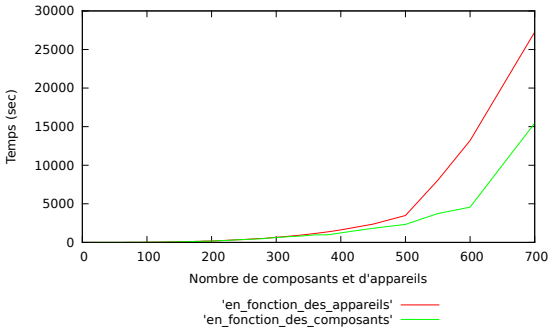
(a) Cardinalité simple



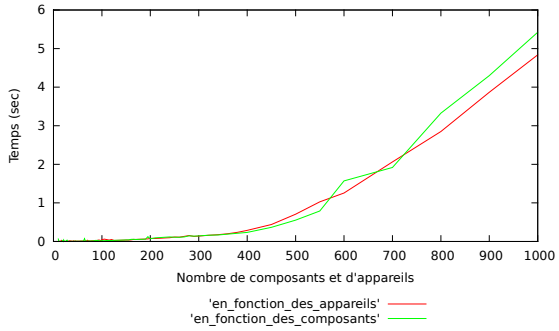
(b) Intervalle



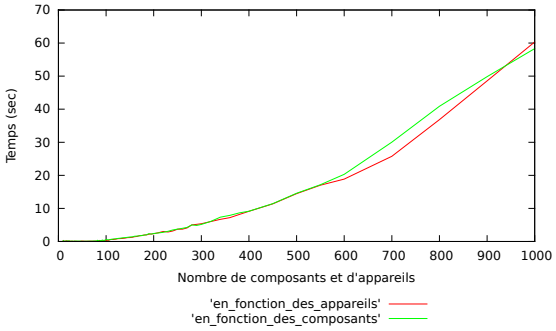
(c) Ratio



(d) `All`



(e) `Each`



(f) Localité au même appareil

Figure 8.13 – Temps de calcul du plan de déploiement en variant le nombre de composants ou le nombre d'appareils

8.4.2 Passage à l'échelle

Le système de déploiement est composé de deux parties : l'une centralisée (le moniteur de déploiement) et l'autre décentralisée (le support local du système de déploiement sur les appareils).

Pour éviter l'effet « goulot d'étranglement » dans la partie centralisée, nous utilisons les serveurs RabbitMQ pour la communication entre les appareils et le moniteur. En effet, différentes instances peuvent être utilisées pour répondre au grand nombre d'appareils. De plus, l'ajout à chaud de nouvelles instances est prévu par le système RabbitMQ, ce qui permet de résister à la montée en charge en cas d'arrivée massive d'appareils.

À l'exécution du système déployé, le système de déploiement repose principalement sur une architecture décentralisée. Cette décentralisation permet de pouvoir réagir localement aux perturbations, et ainsi éviter de remonter au moniteur à chaque fois. Afin de fluidifier la communication, nous pouvons utiliser un étiquetage des messages fourni par RabbitMQ. Ceci permettrait aux superviseurs de s'abonner aux seuls messages concernant leur instance d'échelle, et ainsi avoir leur communication locale. Pour la prise en compte de nouveaux appareils localement, nous pouvons utiliser un protocole de découverte de services, en utilisant le protocole UPnP [UPn].

8.5 Conclusion

Nous avons présenté dans ce chapitre un prototype de notre middleware MuScADeM. Il est construit suivant l'architecture définie dans les chapitres 6 et 7. Il offre un environnement de développement intégré pour le langage dédié MuScADeL, sous la forme d'un plugin Eclipse. Le concepteur peut donc écrire le descripteur MuScADeL correspondant à son application et lancer directement la génération du plan de déploiement depuis Eclipse. Le support local de déploiement, utilisant un framework OSGi, permet aux appareils d'être visibles par le gestionnaire du domaine. Le gestionnaire de domaine tire parti de la scalabilité des instances RabbitMQ pour pouvoir traiter sans délai (sans engorgement) les nombreux appareils avec lequel il communique. Une fois le plan de déploiement généré, le concepteur initie, depuis Eclipse, le déploiement selon le plan initial. Le support local récupère donc le composant, l'installe et l'active. Une prise en compte des appareils entrant dans le domaine est aussi implémentée. Pour chaque nouvel appareil détecté par le gestionnaire de domaine, si cet appareil satisfait les propriétés d'un composant, ce composant est déployé dessus si besoin (c'est-à-dire si des propriétés dynamiques ont été définies).

Nous avons aussi effectué des tests de performance de la génération du plan de déploiement. Ces tests permettent de mettre en évidence certains écarts de temps de résolution entre les différentes propriétés exprimées. L'optimisation du traitement de ces propriétés demande un certain travail d'ingénierie.

Contribution

Le middleware MuScADeM supporte le déploiement de systèmes répartis multi-échelles. Il s'appuie sur le DSL MuScADeL et son environnement de développement intégré, pilote le déploiement initial, et gère le déploiement dynamique. Le plugin Eclipse de MuScADeL permet au concepteur d'exprimer le déploiement de son application et d'utiliser la caractérisation multi-échelle, grâce à Xtext et Xtend. Les bootstraps permettent à l'utilisateur une prise en main simple du système de déploiement et, grâce à l'utilisation d'OSGi, de s'abstraire des problèmes d'hétérogénéité des appareils. À partir du descripteur MuScADeL, l'état du domaine est récupéré et un plan de déploiement est généré au moyen du solveur de contraintes Choco. Ensuite, le plan de déploiement est réalisé, en s'appuyant sur RabbitMQ pour envoyer les commandes et le support local de déploiement pour la gestion locale (avec OSGi). La dynamique du domaine est gérée grâce à l'utilisation de RabbitMQ pour la détection de nouveaux appareils, ou la disparition. L'utilisation de RabbitMQ facilite aussi le passage à l'échelle en nombre d'appareils.

9 Conclusion

9.1 Bilan

Dans cette thèse, nous avons étudié le problème du déploiement des systèmes répartis multi-échelles. Nous en avons identifié les principales exigences en lien avec les propriétés caractéristiques d'hétérogénéité, de nombre, de dynamique et d'ouverture. Alors que le déploiement traditionnel est le plus souvent centralisé et dirigé par un opérateur humain, le déploiement des systèmes répartis multi-échelles doit être décentralisé, automatisé et sensible au contexte. Le rôle d'opérateur de déploiement doit y être joué par une application de niveau middleware : le système de déploiement.

Dans ce travail, nous avons proposé une solution qui s'articule autour de trois contributions principales : un processus de déploiement, un langage de spécification et un middleware pour la réalisation automatique.

Le processus de déploiement des systèmes répartis multi-échelles coordonne un certain nombre d'activités qui concernent la spécification du plan de déploiement, la génération et la réalisation du plan de déploiement initial, et la réalisation du déploiement dynamique. En particulier, il prend en compte l'ouverture du domaine de déploiement. La production du plan de déploiement est centralisée mais sa réalisation (déploiement initial et déploiement dynamique) est décentralisée et contrôlée dans le cadre d'une boucle « autonomique ». Pour cela, le processus s'appuie sur une infrastructure ouverte composée d'un support de spécification (un langage et son éditeur) et d'un middleware pour la mise en œuvre. Pour la sensibilité au contexte, le domaine et son état sont observés au moyen de sondes extraites de la spécification des propriétés du déploiement.

La conception du plan de déploiement est une activité particulière qui demande des moyens d'expression adéquats. Ces moyens sont donnés au concepteur sous la forme d'un langage dédié, MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*), qui offre un haut niveau d'abstraction et permet de s'abstraire des questions de mise en œuvre. Le concepteur du déploiement peut exprimer les contraintes d'exécution des composants ainsi que ses exigences en matière de distribution, de nombre et de dynamique du domaine. Pour spécifier le déploiement sans connaissance exacte du domaine et désigner des parties du domaine, il peut s'appuyer sur les échelles identifiées dans le système multi-échelle. MuScADeL est lié au métamodèle MuSCa de S. Rottenberg [Rottenberg et al., 2014], ce qui permet de contrôler la validité de l'expression des propriétés relatives aux échelles. Le concepteur peut également donner les éléments qui vont permettre l'acquisition d'un état personnalisé du domaine, en particulier il peut définir ses propres sondes. Ces sondes définies en fonction du besoin peuvent supporter des contrôles d'ordre divers : énergie, sécurité, qualité, etc.

C'est cette possibilité de personnalisation qui permet de préserver la généricité du langage et du middleware.

Pour l'automatisation du déploiement, nous avons proposé un middleware réparti, MuScADeM (***Mu**lti**Sc**ale **A**utonomic **De**ployment **M**iddleware*) qui supporte le déploiement initial de manière automatique, de la constitution du domaine et l'acquisition de son état jusqu'à la réalisation du plan de déploiement. L'architecture de MuScADeM comprend un gestionnaire de domaine qui enregistre les entrées et les sorties des appareils, un moniteur de déploiement et un support local de déploiement sur chaque appareil. Le support local est un conteneur de composants, initialement minimal, qui s'enrichit dynamiquement de composants pour la réalisation du déploiement. Il héberge à la fois les composants du système de déploiement et ceux de l'application déployée. Le moniteur de déploiement utilise un solveur de contraintes pour produire un plan de déploiement initial qui est à la fois adapté à l'état du domaine et conforme aux spécifications exprimées. Pour cela, les propriétés spécifiées sont transformées automatiquement en contraintes avant d'être traitées par le solveur.

Une fois le déploiement initial effectué, le système de déploiement doit maintenir en condition opérationnelle le système déployé en présence de perturbations. Pour réagir dynamiquement à ces perturbations, le système de déploiement réagit de manière autonomique. Nous avons identifié et analysé les différentes situations d'adaptation et, pour chacune, proposé une solution. Au vu des conditions d'exécution du système déployé (dynamique et taille du domaine), une architecture centralisée n'est pas envisageable. Nous proposons les éléments d'une architecture décentralisée qui permet une supervision par instance d'échelle et fournit, de manière autonomique, une réponse locale aux situations rencontrées.

Un prototype, qui satisfait l'essentiel des exigences énoncées, a été réalisé conformément à l'architecture définie. Un projet Master 1 ainsi qu'un stage de Master 2 y ont contribué en partie. Ce prototype prouve la faisabilité de nos propositions de processus et d'architecture.

Ce travail a été réalisé dans le cadre du projet INCOME financé par l'Agence Nationale de la Recherche (ANR-11-INFR-009, 2012-2015). Il a fait l'objet de deux articles publiés dans des revues internationales [Boujbel et al., 2014a, Arcangeli et al., 2015], de deux communications dans des manifestations internationales [Boujbel et al., 2013, Boujbel et al., 2014c], de trois communications nationales [Boujbel et al., 2014b, Arcangeli et al., 2014b, Arcangeli et al., 2014c], d'un poster [Kem et al., 2014] et de différents livrables du projet [Arcangeli et al., 2012a, Arcangeli et al., 2013, Arcangeli et al., 2014a].

9.2 Perspectives

Pour terminer, nous discutons dans cette dernière section quelques questions ouvertes et perspectives de ce travail.

9.2.1 Intégration et validation

Au sein du projet INCOME, les tâches d'intégration et de validation sont essentiellement planifiées en 2015. En particulier, MuScADeL et MuScADeM seront utilisés pour déployer une application réelle, à savoir le gestionnaire de contexte multi-échelle développé dans le cadre du projet et, éventuellement, une ou des applications sensibles au contexte qui utilisent le gestionnaire de contexte. Pour l'instant, notre prototype n'a été expérimenté que

sur des cas relativement simples. Nous avons montré que MuScADeM répond à l'essentiel des exigences identifiées dans le chapitre 2. Mais pour certaines nous avons fait des hypothèses simplificatrices, par exemple concernant les infrastructures de Cloud et les réseaux de capteurs. Un important travail d'ingénierie est sans doute nécessaire pour interfacer notre système de déploiement avec les outils de déploiement sur ces infrastructures.

D'autre part, pour la génération du plan de déploiement, si une solution existe, elle est trouvée par le solveur de contraintes. En fait, c'est la première trouvée qui est retenue sans considérer la qualité du plan ni un quelconque optimum. Cette question de la qualité du plan (qui ne faisait pas partie des exigences) pourrait néanmoins être reconsidéré. Par ailleurs, concernant le temps de génération du plan, un travail d'optimisation devrait permettre d'améliorer les performances (par exemple, pour la propriété `All`).

9.2.2 Expression du déploiement

Le langage MuScADeL répond aux principaux besoins que nous avons identifiés pour le déploiement des systèmes répartis multi-échelles. Cependant, il ne permet d'exprimer des propriétés relatives à toutes les activités de déploiement, comme la désactivation ou la désinstallation, ou encore le déploiement incrémental (cf. section 9.2.3). En fonction des besoins, en particulier pour les expérimentations à venir avec des applications réelles, nous prévoyons donc des adaptations de MuScADeL pour permettre l'expression de nouvelles propriétés de déploiement.

9.2.3 Dynamique du domaine et de l'application

Dans ce travail, nous nous sommes focalisés sur la dynamique du domaine pour répondre aux problèmes de connexion et de déconnexion d'appareils (donc sur le déploiement continu). Cependant, nous avons fait abstraction des problèmes de connexité du domaine, c'est-à-dire de la possibilité pour un appareil de sortir puis d'entrer à nouveau dans le domaine ou, pour le domaine, de la possibilité de se fragmenter. Pour cette dernière question, l'approche autonomique pourrait permettre aux « fragments » du domaine de rester opérationnels, mais le problème doit être étudié précisément.

Par ailleurs, la dynamique du système réparti ne se limite pas à celle du domaine. Les applications ont une durée de vie suffisamment longue pour avoir à évoluer dynamiquement par la mise à jour, l'ajout ou le retrait de composants. Si, normalement, la mise à jour n'influe pas sur le plan de déploiement, l'ajout ou le retrait de composants induit par nature une modification du plan. Dans le cas d'un ajout d'un nouveau composant, il faut trouver le « bon » appareil pour l'héberger mais cela peut remettre en cause le plan de déploiement de l'ensemble du système. Le déploiement incrémental doit donc faire l'objet d'une étude approfondie.

9.2.4 Déploiement autonomique

Pour le maintien en condition opérationnelle de l'application, le système de déploiement effectue une supervision et un contrôle décentralisés. Comme nous l'avons montré, dans le cadre de l'architecture hiérarchique par instance d'échelle, des conflits de contrôle peuvent se produire. Une approche à base de négociation pourrait permettre de résoudre ces conflits de manière décentralisée. Pour cela, nous envisageons une conception du support local

de déploiement sous la forme d'un système d'agents autonomes au sein duquel les activités de déploiement seraient distribuées. Parmi ces agents, certains pourraient interagir avec d'autres agents de la même ou d'une autre instance d'échelle et négocier en cas de conflit. En particulier, des agents contrôleurs pourraient coopérer au sein d'un système multi-agent dans le but de régler les conflits. L'approche AMAS (*Adaptive Multi-Agent Systems*) [Gleizes et al., 1999, Capera et al., 2003] développée dans notre équipe pourrait apporter une solution complètement décentralisée et permettre la suppression des superviseurs hiérarchiques.

Bibliographie et Webographie

Bibliographie

[WET, 2003] (2003). *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, 9-11 June 2003, Linz, Austria*. IEEE Computer Society.

[ICA, 2004] (2004). *1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA*. IEEE Computer Society.

[CCG, 2008] (2008). *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), 19-22 May 2008, Lyon, France*. IEEE Computer Society.

[AIN, 2010] (2010). *24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, 20-13 April 2010*. IEEE Computer Society.

[EUR, 2010] (2010). *36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010, Lille, France, September 1-3, 2010*. IEEE.

[ICA, 2010] (2010). *Sixth International Conference on Autonomic and Autonomous Systems, ICAS 2010, Cancun, Mexico, March 7-13, 2010*. IEEE Computer Society.

[NCA, 2012] (2012). *11th IEEE International Symposium on Network Computing and Applications, NCA 2012, Cambridge, MA, USA, August 23-25, 2012*. IEEE Computer Society.

[COM, 2014] (2014). *IEEE 38th Annual International Computers, Software and Applications Conference Workshops, COMPSAC Workshops 2014, Sweden, July 21-25, 2014*. IEEE.

[Arcangeli et al., 2012a] Arcangeli, J.-P., Boujbel, R., Bouzeghoub, A., Camps, V., Chabridon, S., Conan, D., Desprats, T., Guivarch, V., Laborde, R., Lavinal, E., Leriche, S., Oglaza, A., Péninou, A., Rottenberg, S., Taconet, C., and Zaraté, P. (2012a). Multi-scale context management : Concepts, Definitions, and State of the art. ANR INFRA INCOME Multi-Scale Context Management for the Internet of Things deliverable Task 2.1.

[Arcangeli et al., 2013] Arcangeli, J.-P., Boujbel, R., Bouzeghoub, A., Camps, V., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Lavinal, E., Leriche, S., Marie, P., Oglaza, A., Péninou, A., Rottenberg, S., Taconet, C., and Zaraté, P. (2013). Scénarios et exigences. ANR INFRA INCOME INfrastructure de gestion de COntexte Multi-Échelle pour l'Internet des Objets Tâche 2.2.

[Arcangeli et al., 2014a] Arcangeli, J.-P., Boujbel, R., and Leriche, S. (2014a). Infrastructure logicielle de déploiement du gestionnaire de contexte — Version 1. ANR INFRA INCOME INfrastructure de gestion de COntexte Multi-Échelle pour l'Internet des Objets Tâche 5.1.

[Arcangeli et al., 2015] Arcangeli, J.-P., Boujbel, R., and Leriche, S. (2015). Automatic deployment of distributed software systems : Definitions and state of the art. *Journal of Systems and Software*, 103 :198 – 218.

[Arcangeli et al., 2012c] Arcangeli, J.-P., Bouzeghoub, A., Camps, V., Canut, C. M.-F., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Lavinal, E., Leriche, S., Maurel, H., Péninou, A., Taconet, C., and Zaraté, P. (2012c). INCOME - Multi-scale context management for the Internet of Things. In [Paternò et al., 2012], pages 338–347.

[Arcangeli et al., 2014b] Arcangeli, J.-P., Bouzeghoub, A., Camps, V., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Lavinal, E., Leriche, S., Maurel, H., Mbarki, M., Péninou, A., Taconet, C., Zaraté, P., Boujbel, R., Lim, L., Machara Marquez, S., Marie, P., Mignard, C., Oglaza, A., and Rottenberg, S. (2014b). Projet INCOME : INfrastructure de gestion de COntexte Multi-Echelle pour l'Internet des Objets (short paper). In *Conférence Francophone sur les Architectures Logicielles (CAL)*. ENSEEIHT.

[Arcangeli et al., 2014c] Arcangeli, J.-P., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Leriche, S., Lim, L., Taconet, C., Boujbel, R., Machara Marquez, S., Marie, P., and Rottenberg, S. (2014c). Gestion de contexte multi-échelle pour l'Internet des objets (regular paper). In *Journées francophones Mobilité et Ubiquité (UBIMOB)*, http ://www.i3s.unice.fr. Laboratoire i3S.

[Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things : A survey. *Computer Networks*, 54(15) :2787–2805.

[Bailey, 2000] Bailey, E. C. (2000). *Maximum RPM*. SAMS Publishing.

[Boehm et al., 1999] Boehm, B. W., Garlan, D., and Kramer, J., editors (1999). *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*. ACM.

[Boujbel et al., 2013] Boujbel, R., Leriche, S., and Arcangeli, J.-P. (2013). A DSL for multiscale and autonomic software deployment. In Lavazza, L., Oberhauser, R., Martin, A., Hassine, J., Gebhart, M., and Jäntti, M., editors, *International Conference on Software Engineering Advances (ICSEA 2013)*, pages 291–296.

[Boujbel et al., 2014a] Boujbel, R., Leriche, S., and Arcangeli, J.-P. (2014a). A framework for autonomic software deployment of multiscale systems. *International Journal on Adanced Systems*, 7(1 & 2) :353–369.

[Boujbel et al., 2014b] Boujbel, R., Leriche, S., Arcangeli, J.-P., and Kem, O. (2014b). Formalisation de l'expression d'un plan de déploiement autonomique à base de contraintes. In *Journées francophones Mobilité et Ubiquité (UBIMOB 2014)*, http ://www.i3s.unice.fr. Laboratoire i3S.

[Boujbel et al., 2014c] Boujbel, R., Rottenberg, S., Leriche, S., Taconet, C., Arcangeli, J.-P., and Lecocq, C. (2014c). MuScADeL : A Deployment DSL based on a Multiscale Characterization Framework. In [**?**], pages 708–715.

[Brandtzæg et al., 2012] Brandtzæg, E., Parastoo, M., and Mosser, S. (2012). Towards a Domain-Specific Language to Deploy Applications in the Clouds. In *3rd Int. Conf. on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2012)*, pages 213–218. IARIA.

[Briand and Wolf, 2007] Briand, L. C. and Wolf, A. L., editors (2007). *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*.

[Broto et al., 2008] Broto, L., Hagimont, D., Stolf, P., Palma, N. D., and Temate, S. (2008). Autonomic management policy specification in Tune. In [Wainwright and Haddad, 2008], pages 1658–1663.

[Bruneton et al., 2006] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., and Stefani, J.-B. (2006). The FRACTAL component model and its support in Java. *Software : Practice and Experience*, 36(11-12) :1257–1284.

[Budinsky et al., 2008] Budinsky, F., Merks, E., and Steinberg, D. (2008). *Eclipse Modeling Framework 2.0*. Eclipse. Addison Wesley Professional.

[Capera et al., 2003] Capera, D., Georgé, J., Gleizes, M. P., and Glize, P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In [**?**], pages 383–388.

[Carzaniga et al., 1998] Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., van der Hoek, A., and Wolf, A. L. (1998). A characterization framework for software deployment technologies. Technical report, Defense Technical Information Center (DTIC) Document.

[Cassagnes et al., 2009] Cassagnes, C., Roose, P., and Dalmau, M. (2009). Kalimucho : software architecture for limited mobile devices. *ACM SIGBED Review*, 6(3) :12.

[C.H.O.C.O. Team, 2010] C.H.O.C.O. Team (2010). CHOCO : an open source Java constraint programming library. Technical Report 10-02-INFO, Ecole des Mines de Nantes. Last access : June 2014.

[Chu et al., 2011] Chu, W. C., Wong, W. E., Palakal, M. J., and Hung, C.-C., editors (2011). *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*. ACM.

[Crnkovic et al., 2011] Crnkovic, I., Sentilles, S., Vulgarakis, A., and Chaudron, M. R. V. (2011). A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*, 37(5) :593–615.

[Cudennec et al., 2008] Cudennec, L., Antoniu, G., and Bougé, L. (2008). CoRDAGe : Towards Transparent Management of Interactions Between Applications and Resources. In *International Workshop on Scalable Tools for High-End Computing (STHEC 2008)*, pages 13–24.

[Cunin et al., 2005] Cunin, P.-Y., Lestideau, V., and Merle, N. (2005). Orya : A strategy oriented deployment framework. In Dearle, A. and Eisenbach, S., editors, *Component Deployment*, volume 3798 of *Lecture Notes in Computer Science*, pages 177–180. Springer Berlin Heidelberg.

[Dearle, 2007] Dearle, A. (2007). Software Deployment, Past, Present and Future. In [Briand and Wolf, 2007], pages 269–284.

[Dearle et al., 2010] Dearle, A., Kirby, G. N. C., and McCarthy, A. (2010). A middleware framework for constraint-based deployment and autonomic management of distributed applications. *CoRR*, abs/1006.4733.

[Dearle et al., 2004] Dearle, A., Kirby, G. N. C., and McCarthy, A. J. (2004). A framework for constraint-based deployment and autonomic management of distributed applications. In [ICA, 2004], pages 300–301.

[Desertot et al., 2006] Desertot, M., Cervantes, H., and Donsez, D. (2006). FROGi : Fractal Components Deployment over OSGi. In Löwe, W. and Südholt, M., editors, *Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 275–290. Springer.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes multi-agents : vers une intelligence collective*. I.I.A. Informatique intelligence artificielle. InterEditions.

[Ferscha, 2012] Ferscha, A. (2012). 20 years past weiser : What's next ? *IEEE Pervasive Computing*, 11(1) :52–61.

[Flinn, 2012] Flinn, J. (2012). *Cyber Foraging : Bridging Mobile and Cloud Computing*. Synthesis Lectures on Mobile and Pervasive Computing. Morgan & Claypool Publishers.

[Flissi et al., 2008a] Flissi, A., Dubus, J., Dolet, N., and Merle, P. (2008a). Deploying on the grid with deployware. In [CCG, 2008], pages 177–184.

[Ganzha et al., 2014] Ganzha, M., Maciaszek, L. A., and Paprzycki, M., editors (2014). *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014*.

[Giese and Cheng, 2011] Giese, H. and Cheng, B. H. C., editors (2011). *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*. ACM.

[Gleizes et al., 1999] Gleizes, M.-P., Camps, V., and Glize, P. (1999). A Theory of Emergent Computation based on Cooperative Self-organization for Adaptive Artificial Systems. In *Fourth European Congress of Systems Science , Valencia Spain, 20/09/1999-24/09/1999*. Ferrer Figueras, L., editor,.

[Goldsack et al., 2009] Goldsack, P., Guijarro, J., Loughran, S., Coles, A. N., Farrell, A., Lain, A., Murray, P., and Toft, P. (2009). The SmartFrog configuration management framework. *Operating Systems Review*, 43(1) :16–25.

[Guidec et al., 2010] Guidec, F., Sommer, N. L., and Mahéo, Y. (2010). Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *International Journal of Handheld Computing Research*, 1(1) :24–42.

[Hall et al., 1999] Hall, R. S., Heimbigner, D., and Wolf, A. L. (1999). A Cooperative Approach to Support Software Deployment Using the Software Dock. In [Boehm et al., 1999], pages 174–183.

[Haller and Odersky, 2006] Haller, P. and Odersky, M. (2006). Event-Based Programming Without Inversion of Control. In Lightfoot, D. and Szyperski, C., editors, *Modular Programming Languages*, volume 4228, chapter Lecture Notes in Computer Science, pages 4–22. Springer Berlin Heidelberg.

[Heydarnoori, 2008] Heydarnoori, A. (2008). Deploying component-based applications : Tools and techniques. In Lee, R., editor, *Software Engineering Research, Management and Applications*, volume 253 of *Studies in Computational Intelligence*, pages 29–42. Springer-Verlag, Prague, Czech Republic.

[Horn, 2001] Horn, P. (2001). Autonomic computing : IBM's Perspective on the State of Information Technology.

[Horré et al., 2011] Horré, W., Michiels, S., Joosen, W., and Hughes, D. (2011). Advanced sensor network software deployment using application-level quality goals. *Journal of Software*, 6(4) :528–535.

[Hughes et al., 2012] Hughes, D., Thoelen, K., Maerien, J., Matthys, N., Horré, W., del Cid, J., Huygens, C., Michiels, S., and Joosen, W. (2012). Looci : The loosely-coupled component infrastructure. In [NCA, 2012], pages 236–243.

[ISO 9000:2005, 2009] ISO 9000:2005 (2009). Quality management systems – Fundamentals and vocabulary.

[Juve and Deelman, 2011a] Juve, G. and Deelman, E. (2011a). Automating Application Deployment in Infrastructure Clouds. In *IEEE Third Int. Conf. on Cloud Computing Technology and Science (CloudCom 2011)*, pages 658–665.

[Kem et al., 2014] Kem, O., Boujbel, R., and Leriche, S. (2014). Récupération de l'état d'un domaine de déploiement (short paper and demo). In *Journées francophones Mobilité et Ubiquité (UBIMOB 2014)*, http ://www.i3s.unice.fr. Laboratoire i3S.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1) :41–50.

[Kessis et al., 2009] Kessis, M., Roncancio, C., and Lefebvre, A. (2009). DASIMA : A flexible management middleware in multi-scale contexts. In *6th Int. Conf. on Information Technology : New Generations (ITNG '09)*, pages 1390–1396.

[Liu, 2011] Liu, H. (2011). Rapid application configuration in amazon cloud using configurable virtual appliances. In [Chu et al., 2011], pages 147–154.

[Louberry et al., 2011b] Louberry, C., Roose, P., and Dalmau, M. (2011b). Kalimucho : Contextual Deployment for QoS Management. In Felber, P. and Rouvoy, R., editors, *Distributed Applications and Interoperable Systems (DAIS 2011)*, pages 43–56.

[Manzoor and Nefti, 2008] Manzoor, U. and Nefti, S. (2008). Silent Unattended Installation Package Manager–SUIPM. In Mohammadian, M., editor, *Int. Conf. CIMCA/IAWTIC/ISE*, pages 55–60. IEEE Computer Society.

[Manzoor and Nefti, 2010] Manzoor, U. and Nefti, S. (2010). QUIET : A Methodology for Autonomous Software Deployment using Mobile Agents. *J. Network and Computer Applications*, 33(6) :696–706.

[Marrón et al., 2006] Marrón, P. J., Gauger, M., Lachenmann, A., Minder, D., Saukh, O., and Rothermel, K. (2006). Flexcup : A flexible and efficient code update mechanism for sensor networks. In [Römer et al., 2006], pages 212–227.

[Matougui and Leriche, 2012] Matougui, M. E. A. and Leriche, S. (2012). A middleware architecture for autonomic software deployment. In *The 7th Int. Conf. on Systems and Networks Communications (ICSNC'12)*, pages 13–20, Lisbon, Portugal. XPS. 12619 12619.

[OSGi Alliance, 2009] OSGi Alliance (2009). OSGi Service Platform Core Specification, Release 3. Version 4.2. Technical report.

[Paternò et al., 2012] Paternò, F., Ruyter, B. E. R. d., Markopoulos, P., Santoro, C., Loenen, E. v., and Luyten, K., editors (2012). *Ambient Intelligence - Third International Joint Conference, AmI 2012, Pisa, Italy, November 13-15, 2012. Proceedings*, volume 7683 of *Lecture Notes in Computer Science*. Springer.

[Rellermeyer et al., 2007] Rellermeyer, J. S., Alonso, G., and Roscoe, T. (2007). R-OSGi : Distributed Applications Through Software Modularization. In Cerqueira, R. and Campbell, R. H., editors, *8th International Middleware Conference*, volume 4834 of *Lecture Notes in Computer Science*, pages 1–20. Springer.

[Römer et al., 2006] Römer, K., Karl, H., and Mattern, F., editors (2006). *Wireless Sensor Networks, Third European Workshop, EWSN 2006, Zurich, Switzerland, February 13-15, 2006, Proceedings*, volume 3868 of *Lecture Notes in Computer Science*. Springer.

[Rottenberg, 2015] Rottenberg, S. (2015). *Modèles, méthodes et outils pour les systèmes répartis multi-échelles*. PhD thesis, Télécom SudParis. Version provisoire.

[Rottenberg et al., 2014] Rottenberg, S., Leriche, S., Taconet, C., Lecocq, C., and Desprats, T. (2014). MuSCa : A multiscale characterization framework for complex distributed systems. In [Ganzha et al., 2014], pages 1657–1665.

[Sabharwal, 2006] Sabharwal, R. (2006). Grid Infrastructure Deployment using SmartFrog Technology. In *Int. Conf. on Networking and Services (ICNS 2006)*, page 73. IEEE Computer Society.

[Satyanarayanan et al., 2009] Satyanarayanan, M., Bahl, P., Cáceres, R., and Davies, N. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4) :14–23.

[Sledziewski et al., 2010] Sledziewski, K., Bordbar, B., and Anane, R. (2010). A DSL-based approach to software development and deployment on cloud. In [AIN, 2010], pages 414–421.

[Strembeck and Zdun, 2009] Strembeck, M. and Zdun, U. (2009). An approach for the systematic development of domain-specific languages. *Software : Practice and Experience*, 39(15) :1253–1292.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

[Tolvanen and Kelly, 2010] Tolvanen, J.-P. and Kelly, S. (2010). Integrating models with domain-specific modeling languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 10–1, New York, NY, USA. ACM.

[Toure et al., 2010] Toure, M., Stolf, P., Hagimont, D., and Broto, L. (2010). Large scale deployment. In [ICA, 2010], pages 78–83.

[van der Burg and Dolstra, 2010a] van der Burg, S. and Dolstra, E. (2010a). Automated Deployment of a Heterogeneous Service-Oriented System. In [EUR, 2010], pages 183–190.

[van der Burg and Dolstra, 2011] van der Burg, S. and Dolstra, E. (2011). A Self-Adaptive Deployment Framework for Service-Oriented Systems. In [Giese and Cheng, 2011], pages 208–217.

[van der Burg and Dolstra, 2014] van der Burg, S. and Dolstra, E. (2014). Disnix : A Toolset for Distributed Deployment. *Science of Computer Programming*, 79 :52–69.

[Van Deursen et al., 2000] Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages : An annotated bibliography. *ACM Sigplan Notices*, 35(6) :26–36.

[van Steen et al., 2012] van Steen, M., Pierre, G., and Voulgaris, S. (2012). Challenges in very large distributed systems. *Journal of Internet Services and Applications*, 3(1) :59–66.

[VMware Inc., 2008] VMware Inc. (2008). Building the Virtualized Enterprise with VMware Infrastructure. `http://www.vmware.com/pdf/vmware_infrastructure_wp.pdf`. White paper.

[Wainwright and Haddad, 2008] Wainwright, R. L. and Haddad, H., editors (2008). ACM.

[Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific American*.

[Weiser, 1999] Weiser, M. (1999). The computer for the 21st century. *Mobile Computing and Communications Review*, 3(3) :3–11.

[Zheng et al., 2006] Zheng, D., Wang, J., Han, W., Jia, Y., and Zou, P. (2006). Towards A Context-Aware Middleware for Deploying Component-Based Applications in Pervasive Computing. In *5th Int. Conf. on Grid and Cooperative Computing (GCC 2006)*, pages 454–457. IEEE Computer Society.

[Zheng et al., 2007] Zheng, D., Wang, J., Jia, Y., Han, W., and Zou, P. (2007). Deployment of Context-Aware Component-Based Applications Based on Middleware. In Indulska, J., Ma, J., Yang, L. T., Ungerer, T., and Cao, J., editors, *Ubiquitous Intelligence and Computing (UIC 2007)*, volume 4611 of *LNCS*, pages 908–918. Springer.

Webographie

[ADA] Adage : An Automatic Deployment Tool. http://adage.gforge.inria.fr, 2007. Last access in november 2014.

[AMQ] AMQP. http://www.amqp.org. Last access in november 2014.

[Fel] Apache Felix - Welcome to Apache Felix. https://felix.apache.org. Last access in november 2014.

[Gso] google-gson - A Java library to convert JSON to Java objects and vice-versa - Google Project Hosting. http://code.google.com/p/google-gson/. Last access in november 2014.

[INC] The INCOME project. www.anr-income.fr, 2012. Last access in november 2014.

[Jet] Apache Felix - Apache Felix HTTP Service. http://felix.apache.org/documentation/subprojects/apache-felix-http-service.html. Last access in november 2014.

[jOp] jOpt, Java OPL implementation. http://jopt.sourceforge.net/opl.php. Last access in november 2014.

[JSO] JSON. http://www.json.org. Last access in november 2014.

[KS] Krzysztof Kuchcinski and Radoslaw Szymanek. JaCoP - Java constraint programming solver. http://jacop.osolpro.com. Last access in november 2014.

[Obj1] Object Management Group. Object Management Group. http://www.omg.org. Last access in november 2014.

[Obj2] Object Management Group. Corba Component Model (CCM). http://www.omg.org/spec/CCM, 2006. Last access in november 2014.

[Obj3] Object Management Group. Deployment and Configuration of Component-based Distributed Applications Specification, Version 4.0. http://www.omg.org/spec/DEPL, 2006. Last access in november 2014.

[Ora2] Oracle. JavaBeans Tutorial. http://docs.oracle.com/javase/tutorial/javabeans/index.html, 2013. Last access in november 2014.

[ort] or-tools, operations research tools developed at Google. https://code.google.com/p/or-tools. Last access in november 2014.

[OW2] OW2 Consortium. The FRACTAL project. http://fractal.ow2.org/documentation.html, 2009. Last access in november 2014.

[Rab] RabbitMQ : Messaging that just work. http://www.rabbitmq.com. Last access in november 2014.

[RES] REST. http://www.xfront.com/REST-Web-Services.html. Last access in november 2014.

[SAR] SARAH - Delay-Tolerant Distributed Services for Mobile Ad Hoc Networks. http://www-valoria.univ-ubs.fr/SARAH, 2005. Last access in 2014.

[SEL] The SELFWARE project. http://sardes.inrialpes.fr/~boyer/selfware, 2005. Last access in november 2014.

[Sig] Technologie. http://www.sigfox.com/fr/#!/technology. Last access in november 2014.

[SPE] SPEM 2.0. http://www.omg.org/spec/SPEM/2.0/, 2008. Last access in november 2014.

[Tam1] Naoyuki Tamura. Copris : Constraint Programming in Scala. http://bach.istc.kobe-u.ac.jp/copris. Last access in november 2014.

[Tam2] Naoyuki Tamura. Cream : Class library for constraint programming in Java. http://bach.istc.kobe-u.ac.jp/cream. Last access in november 2014.

[UPn] UPnP Forum. UPnP Device Architecture. http://www.upnp.org, 2008. Last access in november 2014.

[USE] UseNet - Ubiquitous M2M Service Networks. https://itea3.org/project/usenet.html, 2007. Last access in november 2014.

[Wei] Mark Weiser. Nomadic Issues in Ubiquitous Computing. http://www.ubiq.com/hypertext/weiser/NomadicInteractive/, 1996. Slides presented at the NOMADIC'96 Conference, last access in november 2014.

[XMP] The XMPP Standards Foundation. http://xmpp.org. Last access in november 2014.

[Xte1] Xtend - Modernized Java. http://www.eclipse.org/xtend. Last access in november 2014.

[Xte2] Xtext - Language Development made Easy ! http://www.eclipse.org/Xtext/. Last access in november 2014.

Annexes

A Descripteurs MuScADeL

Ce chapitre décrit le code complet du descripteur de l'exemple donné dans le chapitre 5. L'exemple est composé de cinq fichiers :

- *base.musc* : les définitions des sondes basiques, ce fichier est fourni par le middleware ;
- *probes.musc* : les définitions des sondes définies par le concepteur ;
- *bcriteria.musc* : les définitions des critères ;
- *components.musc* : les définitions des composants ;
- *msprobes.musc* : les définitions des sondes multi-échelles ;
- *deployment.musc* : la définition du déploiement, c'est-à-dire les exigences du concepteur.

```
1  Probe OS {
2    ProbeInterface fr.enac.lii.basicprobes.OSProbe
3  }

5  Probe Java {
6    ProbeInterface fr.enac.lii.basicprobes.JavaProbe
7  }

9  Probe CPU {
10   ProbeInterface fr.enac.lii.basicprobes.CPUProbe
11 }

13 Probe HD {
14   ProbeInterface fr.enac.lii.basicprobes.HDProbe
15 }

17 Probe JVMMemory {
18   ProbeInterface fr.enac.lii.basicprobes.JVMMemoryProbe
19 }

21 Probe Network {
22   ProbeInterface fr.enac.lii.basicprobes.IPProbe
23 }

25 Probe Locale {
26   ProbeInterface fr.enac.lii.basicprobes.LocaleProbe
27 }

29 Probe RAM {
30   ProbeInterface fr.enac.lii.basicprobes.PhysicalMemoryProbe
31 }

33 Probe Screen {
34   ProbeInterface fr.enac.lii.basicprobes.ScreenProbe
```

```
35 }

37 Probe Peripheral {
38   ProbeInterface fr.enac.lii.basicprobes.PeripheralProbe
39 }
```

Listing 1 – Fichier *base.musc*

```
1  Probe WiFi {
2      ProbeInterface fr.irit.wifi.DImpl
3      URL "http://irit.fr/INCOME/wifi.jar"
4  }

6  Probe SigFox {
7      ProbeInterface fr.irit.wifi.DImpl
8      URL "http://irit.fr/INCOME/sigfox.jar"
9  }

11 Probe GPS {
12     ProbeInterface fr.irit.GPS.DImpl
13     URL "http://irit.fr/INCOME/gps.jar"
14 }
```

Listing 2 – Fichier *probes.musc*

```
1  Include "base.musc"
2  Include "probes.musc"

4  BCriterion SigFoxActive { SigFox Exists;  SigFox  Active; }

6  BCriterion WifiActive { WiFi Active ; }

8  BCriterion CPURAM { CPU.proc > 1;  RAM.free > 1; }

10 BCriterion Screen { Screen.size > 4; }

12 BCriterion RAM80 { RAM.free > 1; }

14 BCriterion LinuxOrAndroid {
15     OS.name = "Linux"
16     | OS.name = "Android";
17 }
```

Listing 3 – Fichier *bcriteria.musc*

```
1  Include "bcriteria.musc"

3  Component Bike {
4      Version 1
5      URL "http://income.fr/4ME/bike.jar"
6  }

8  Component BikeAvail {
9      Version 1
10     URL "http://income.fr/4ME/bikeAvail/jar"
11     DeploymentInterface fr.enac.plop.DIimpl
12 }

14 Component GUI {
15     Version 1
16     URL "http://income.fr/4ME/gui.jar"
```

```
17      Constraint RAM80
18      InitOnly Constraint Screen
19  }

21  Component IM {
22      Version 1
23      URL "http://income.fr/4ME/im.jar"
24      Dependency GUI
25      InitOnly Constraint RAM80
26  }

28  Component IMHist {
29      Version 1
30      URL "http://income.fr/4ME/im_hist.jar"
31  }

33  Component Stat {
34      Version 2
35      URL "http://income.fr/4ME/stat.jar"
36      Constraint CPURAM
37  }

39  Component RoutePlanner {
40      Version 1
41      URL "http://income.fr/4ME/routeplanner.jar"
42  }
```

Listing 4 – Fichier *components.musc*

```
1   MultiScaleProbe Device {
2     MultiScaleProbeInterface eu.telecomsudparis.DeviceProbeImpl
3     URL "http://it-sudparis.eu/INCOME/DeviceProbe.jar"
4   }

6   MultiScaleProbe Administration {
7     MultiScaleProbeInterface eu.telecomsudparis.AdminProbeImpl
8     URL "http://it-sudparis.eu/INCOME/AdminProbe.jar"
9   }

11  MultiScaleProbe User {
12    MultiScaleProbeInterface eu.telecomsudparis.UserProbeImpl
13    URL "http://it-sudparis.eu/INCOME/UserProbe.jar"
14  }Probe Piou {
15      ProbeInterface fr.irit.arduino.DIimpl
16      URL "http://irit.fr/INCOME/arduinoProbe.jar"
17  }
```

Listing 5 – Fichier *msprobes.musc*

```
1  Include "base.musc"
2  Include "bcriteria.musc"
3  Include "msprobes.musc"
4  Include "components.musc"

6  Deployment {
7      AllHosts LinuxOrAndroid;

9      Bike @ All, Administration.Level.Service("Toulouse.SharingBikes"),
10             Device.Type.Cloudlet;
11     BikeAvail @ 1/5 Bike, SigFoxActive;
12     Stat @ 10..30, Device.Type.Cloudlet;
13     GUI @ All, Device.Types.Smartphone;
14     IM @ All, Device.Type.Smartphone,
15                  User.NumberOfUsers.Group;
16     IMHist @ Device.Type.Smartphone,
17                       Each User.NumberOfUsers.Group;
18     RoutePlanner @ 2, SameValue Administration.Level.Service(Bike),
19                      Device.StorageCapacity.GigaBytes;
20 }
```

Listing 6 – Fichier *deployment.musc*

B Grammaire EBNF de MuScADeL

Ce chapitre présente la grammaire EBNF de MuScADeL. Elle est aussi accessible à l'adresse suivante : http://anr-income.fr/ebnf-muscadel.html.

⟨*root*⟩ ::= ⟨*muscadel-element*⟩* ⟨*component*⟩ ⟨*muscadel-element*⟩* ⟨*deployment*⟩

⟨*muscadel-element*⟩ ::= ⟨*include*⟩
| ⟨*probe*⟩
| ⟨*bcriterion*⟩
| ⟨*component*⟩
| ⟨*msprobe*⟩

⟨*include*⟩ ::= 'Include' '"' ⟨*file-id*⟩ '"'

⟨*probe*⟩ ::= 'Probe' ⟨*probe-id*⟩ '{'
'ProbeInterface' ⟨*interface*⟩
'URL' ⟨*string*⟩
'}'

⟨*probe-id*⟩ ::= ⟨*id*⟩

⟨*interface*⟩ ::= ⟨*interface-id*⟩ ('.' ⟨*interface-id*⟩)*

⟨*interface-id*⟩ ::= ⟨*id*⟩

⟨*bcriterion*⟩ ::= 'BCriterion' ⟨*bcriterion-id*⟩ '{'
⟨*condition-disj*⟩
(';' ⟨*condition-disj*⟩)*
';' ?
'}'

⟨*condition-disj*⟩ ::= ⟨*condition*⟩ ('|' ⟨*condition*⟩)*

⟨*bcriterion-id*⟩ ::= ⟨*id*⟩

⟨*condition*⟩ ::= ⟨*probe-id*⟩ 'Exists'
| ⟨*probe-id*⟩ 'Active'
| ⟨*probe-id*⟩ '.' ⟨*probe-meth-id*⟩ ⟨*comp*⟩ ⟨*probe-value*⟩

⟨*probe-meth-id*⟩ : := ⟨*id*⟩

⟨*probe-value*⟩ = ⟨*int*⟩
| ⟨*string*⟩

⟨*comp*⟩ : := '<' | '>' | '<=' | '>=' | '='

⟨*component*⟩ : := 'Component' ⟨*component-id*⟩ '{'
'Version' ⟨*int*⟩
'URL' ⟨*string*⟩
('DeploymentInterface' ⟨*interface*⟩)?
('Dependency' ⟨*component-id-list*⟩)?
('InitOnly'? 'Constraint' ⟨*bcriterion-id*⟩+)*
'}'

⟨*component-id*⟩ : := ⟨*id*⟩

⟨*msprobe*⟩ : := 'MultiScaleProbe' ⟨*msprobe-id*⟩ '{'
'MultiScaleProbeInterface' ⟨*interface*⟩
'URL' ⟨*string*⟩
'}'

⟨*ms-probe-id*⟩ : := ⟨*id*⟩

⟨*deployment*⟩ : := 'Deployment' '{'
(⟨*allhost-requirement*⟩ ';')?
⟨*deployment-requirement*⟩
(';' ⟨*deployment-requirement*⟩)*
';'?
'}'

⟨*allhost-requirement*⟩ : := 'AllHosts' ⟨*bcriterion-id*⟩+

⟨*deployment-requirement*⟩ : := ⟨*component-id*⟩ '@' ⟨*requirement-rhs*⟩ (',' ⟨*requirement-rhs*⟩)*

⟨*requirement-rhs*⟩ : := ⟨*location*⟩
| ⟨*quantifier*⟩
| ⟨*bcriterion-id*⟩
| ⟨*mscriterion*⟩

⟨*mscriterion*⟩ : := ⟨*scale*⟩
| 'Each' ⟨*scale*⟩
| 'SameValue' ⟨*ms-expr*⟩
| 'DifferentValue' ⟨*ms-expr*⟩

⟨*ms-expr*⟩ : := ⟨*scale-instance-m1*⟩
| ⟨*scale-instance-m0*⟩

⟨*dim*⟩ : := ⟨*vp-id*⟩ '.' ⟨*dim-id*⟩

⟨*scale*⟩ : := ⟨*dim*⟩ '.' ⟨*sc-id*⟩

⟨*vp-id*⟩ ::= ⟨*id*⟩

⟨*dim-id*⟩ ::= ⟨*id*⟩

⟨*sc-id*⟩ ::= ⟨*id*⟩

⟨*scale-instance-m1*⟩ ::= ⟨*dim*⟩ '(' ⟨*component-id*⟩ ')'

⟨*scale-instance-m0*⟩ ::= ⟨*scale*⟩ '(' ⟨*comp-or-str*⟩ ')'

⟨*location*⟩ ::= ⟨*ipv4*⟩
| ⟨*hostname*⟩

⟨*quantifier*⟩ ::= ⟨*val*⟩
| ⟨*interval*⟩
| 'All'
| ratio

⟨*ratio*⟩ ::= ⟨*int*⟩ '/' ⟨*int*⟩ ⟨*component-id*⟩

⟨*comp-or-str*⟩ ::= ⟨*component-id*⟩
| ⟨*string*⟩

⟨*ipv4*⟩ ::= ⟨*int*⟩ '.' ⟨*int*⟩ '.' ⟨*int*⟩ '.' ⟨*int*⟩

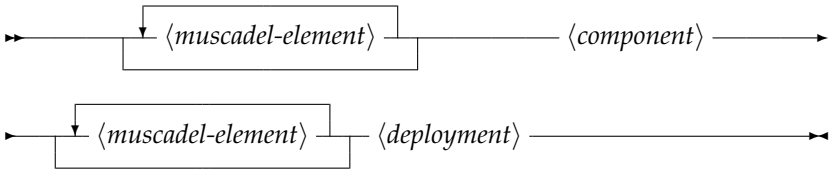
⟨*hostname*⟩ ::= ⟨*string*⟩

⟨*val*⟩ ::= ⟨*int*⟩

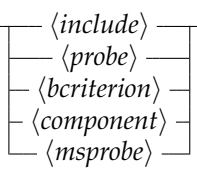
⟨*interval*⟩ ::= ⟨*int*⟩ '..' ⟨*int*⟩

Quelques diagrammes de syntaxe

⟨*root*⟩ : :=



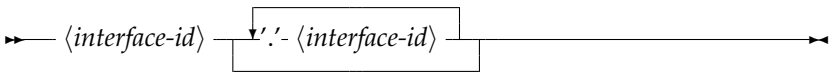
⟨*muscadel-element*⟩ : :=



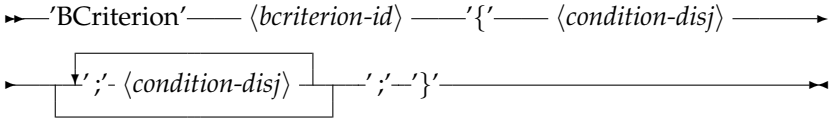
⟨*include*⟩ : :=

⟨*probe*⟩ : :=

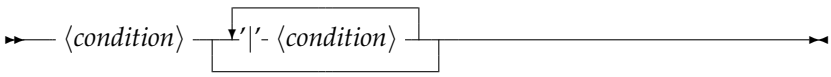
⟨*interface*⟩ : :=



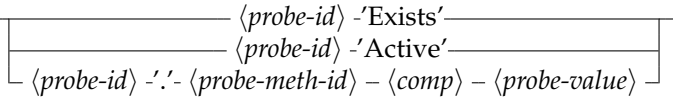
⟨*bcriterion*⟩ : :=



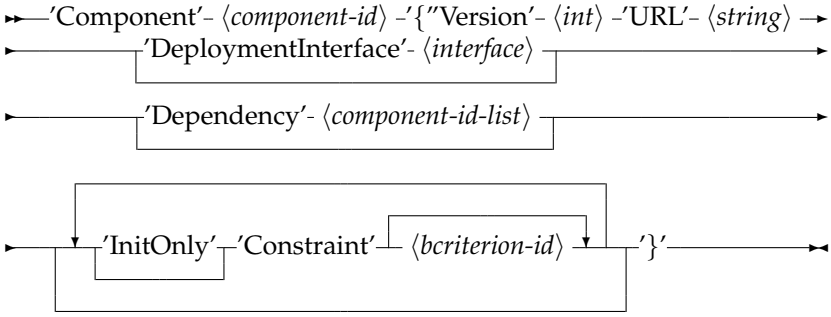
⟨*condition-disj*⟩ : :=



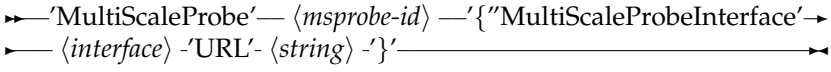
⟨*condition*⟩ : :=



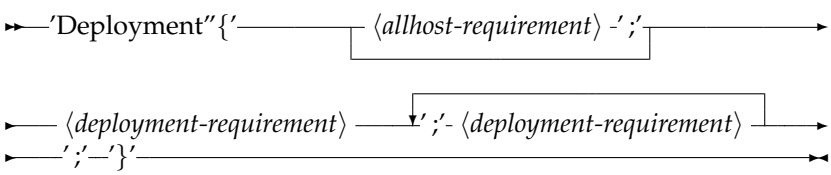
⟨*component*⟩ : :=



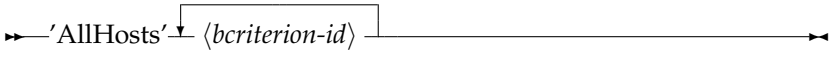
⟨*msprobe*⟩ : :=



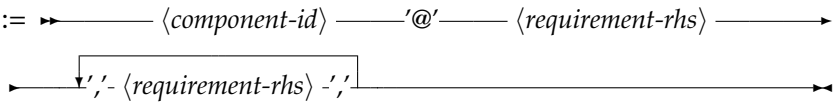
⟨*deployment*⟩ : :=



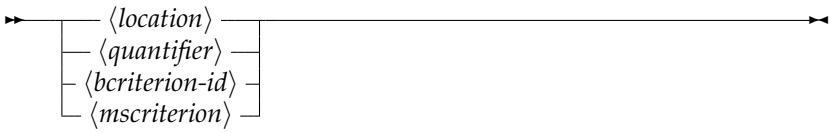
⟨*allhost-requirement*⟩ : :=



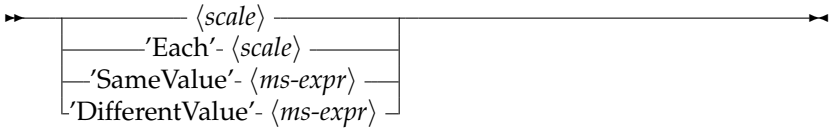
⟨*deployment-requirement*⟩ : :=



⟨*requirement-rhs*⟩ : :=



⟨*mscriterion*⟩ : :=



⟨*ms-expr*⟩ : :=

⟨*dim*⟩ : := ⟨*vp-id*⟩ -'.'- ⟨*dim-id*⟩

⟨*scale*⟩ : := ⟨*dim*⟩ -'.'- ⟨*sc-di*⟩

⟨*scale-instance-m1*⟩ : := ⟨*dim*⟩ -'('- ⟨*component-id*⟩ -')'

⟨*scale-instance-m0*⟩ : := ⟨*scale*⟩ -'('- ⟨*comp-or-str*⟩ -')'

⟨*location*⟩ : :=

⟨*quantifier*⟩ : :=

⟨*ratio*⟩ : := ⟨*int*⟩ -'/'- ⟨*int*⟩ – ⟨*component-id*⟩

C Exigences de la solution de déploiement de systèmes répartis multi-échelles

Dans cette annexe, nous organisons les exigences de la solution de déploiement. Ces exigences sont citées dans le chapitre 8 afin de valider notre solution de déploiement. Les exigences s'organisent selon le cadre d'analyse que nous avons fourni à la section 2.3.1.

L'application

EX 1. Le système de déploiement doit déployer une application (répartie) à base de composants logiciels (c'est-à-dire un « système » de composants logiciels répartis)

EX 2. Les composants déployés doivent être conformes à un modèle de référence (donc homogènes)

EX 3. La solution doit permettre de déployer différents types de composants

EX 4. La solution doit permettre de déployer un grand nombre de composants (ordre de grandeur : plusieurs centaines)

EX 5. La solution doit supporter le déploiement en mode push

EX 6. La solution doit supporter le déploiement d'applications ouvertes, c'est-à-dire le déploiement incrémental de composants (concrètement, on doit pouvoir ajouter ou retirer des composants dynamiquement à l'application)

EX 7. La solution doit permettre au concepteur d'exprimer les propriétés attendues pour l'application à déployer, relativement à la diversité, au nombre, à la dynamique

EX 8. La solution doit permettre de réaliser et de maintenir de manière permanente un plan de déploiement conforme aux propriétés (exprimées par le concepteur) relatives à l'application

Le domaine de déploiement

EX 9. La solution doit supporter le déploiement dans/sur un système réparti multi-échelle (domaine de déploiement)

EX 10. La solution doit permettre le déploiement sur des appareils différents et des réseaux

hétérogènes

EX 11. Le système de déploiement doit prendre la forme d'un middleware qui supportera les interactions distantes

EX 12. La solution doit permettre de déployer l'application sur un grand nombre d'appareils (ordre de grandeur : plusieurs milliers)

EX 13. La solution doit automatiser le déploiement

EX 14. Un programme d'amorce (bootstrap) doit être installé sur tous les appareils du domaine

EX 15. Pour « passer à l'échelle », la solution doit décentraliser les opérations de déploiement

EX 16. La solution doit permettre le déploiement dans des domaines instables, c'est-à-dire dont l'état (disponibilité et qualité des ressources et des services) varie dynamiquement

EX 17. Le déploiement doit être sensible l'état du domaine

EX 18. La solution doit réaliser un plan de déploiement initial adapté à l'état initial du domaine de déploiement

EX 19. Au-delà d'un plan de déploiement « en dur », la solution doit permettre au concepteur d'exprimer des propriétés de déploiement (spécification)

EX 20. Le plan de déploiement initial doit être « calculé » à partir des spécifications et de l'état initial du domaine

EX 21. Le système de déploiement doit adapter dynamiquement le plan de déploiement aux changements d'état du domaine (déploiement continu)

EX 22. Le système de déploiement doit connaître les propriétés de déploiement à préserver et l'état du domaine

EX 23. Pour « passer à l'échelle », le système de déploiement doit traiter localement (le plus localement possible) les problèmes d'adaptation

EX 24. Le système de déploiement doit opérer sur des appareils dont il n'est pas administrateur

EX 25. Le système de déploiement doit opérer sur les réseaux de capteurs, par l'intermédiaire de machines passerelles

EX 26. Le système de déploiement doit opérer sur les machines du Cloud par l'intermédiaire de machines passerelles

EX 27. La solution pourra éventuellement permettre le déploiement de composants sur des machines qui ne sont pas identifiées comme faisant partie du domaine (incompatibilité technique) par l'intermédiaire d'une machine passerelle (qui, elle, fait partie du domaine)

EX 28. La solution doit permettre au concepteur d'exprimer les propriétés du déploiement en fonction du domaine de déploiement

EX 29. La solution doit permettre de réaliser et de maintenir de manière permanente un plan de déploiement conforme aux propriétés (exprimées par le concepteur) relatives au domaine de déploiement

Conception et réalisation du déploiement

Conception du déploiement

EX 30. La solution doit être utilisable par un humain, dans un rôle de concepteur de déploiement, qui n'est pas spécialiste des techniques de déploiement mais de l'application à déployer

EX 31. La solution doit offrir un langage dédié (DSL) dans lequel le concepteur peut exprimer la spécification du plan de déploiement

EX 32. Le concepteur utilisera un éditeur syntaxique dédié qui effectuera un certain nombre de contrôles sur la spécification

EX 33. Le DSL doit permettre d'exprimer avec abstraction les types de composants (code, contraintes d'exécution, etc.), les choix de conception sous la forme de propriétés du déploiement et les informations significatives de l'état du domaine ou comment les acquérir

Génération du plan de déploiement

EX 34. Le plan doit être calculé automatiquement en fonction des propriétés spécifiées et de l'état courant du domaine

EX 35. L'état courant du domaine doit être construit automatiquement à partir des données fournies par les sondes définies dans la spécification

EX 36. Le plan de déploiement doit être calculé « en ligne » (« à chaud »)

Réalisation du déploiement initial

EX 37. Le plan de déploiement initial doit être réalisé automatiquement (installation et activation des composants)

Réalisation du déploiement dynamique

EX 38. Le système de déploiement doit adapter le plan de déploiement en fonctions des variations de l'état du domaine, sans intervention humaine

EX 39. Le système de déploiement doit acquérir des informations de qualité (pertinence, fraîcheur, etc.) sur l'état du domaine

EX 40. Le système de déploiement doit décider des adaptations à effectuer en fonction des propriétés de déploiement à préserver et de l'état acquis du domaine

EX 41. Le système de déploiement doit réaliser les adaptations décidées

Les activités de déploiement

EX 42. La solution doit permettre la mise à disposition de l'application aux utilisateurs, c'est-à-dire supporter toutes les activités qui précèdent l'exécution de l'application : conception du plan de déploiement et réalisation du plan de déploiement initial

EX 43. Le déploiement doit s'effectuer dans le cadre d'un processus automatisé

EX 44. Le processus de déploiement doit d'abord enchaîner les phases de spécification, de construction du plan, puis de réalisation (installation et activation de l'application)

EX 45. Un middleware doit supporter le processus de déploiement

Deployment of distributed multiscale software systems : process, language, and middleware

Abstract

Due to increased connected objects, multiscale systems are more and more widespread. Those systems are highly distributed, heterogeneous, dynamic and open. They can be composed of hundreds of software components deployed into thousands of devices.

Deployment of software systems is a complex post-production process that consists in making software available for use and then keeping it operational. For multiscale systems, deployment plan expression just as deployment realization and management are tasks impossible for a human stakeholder because of heterogeneity, dynamics, number, and also because the deployment domain is not necessarily known in advance.

The purpose of this thesis is to study and propose solutions for the deployment of distributed multiscale software systems. Firstly, we provide an up-to-date terminology and definitions related to software deployment, plus a state of the art on automatic deployment of distributed software systems. The rest of the contribution lies in the proposition of :

— a complete process for autonomic deployment of multiscale systems ;
— a domain specific language, MuScADeL, which simplifies the deployment conceptor task and allows the expression of deployment properties such as informations for the domain state probing ;
— and a middleware, MuScADeM, which insures the automatic generation of a deployment plan according the domain state, its realization and finally the maintenance in an operational condition of the system.

Raja BOUJBEL

DÉPLOIEMENT DE SYSTÈMES RÉPARTIS MULTI-ÉCHELLES : PROCESSUS, LANGAGE ET OUTILS INTERGICIELS

Directeurs de thèse :

Jean-Paul ARCANGELI, *Université de Toulouse, UPS*

Sébastien LERICHE, *Université de Toulouse, ENAC*

Soutenue le 30 janvier 2015 à l'Institut de Recherche en Informatique de Toulouse

---

Résumé

Avec la multiplication des objets connectés, les systèmes multi-échelles sont de plus en plus répandus. Ces systèmes sont fortement répartis, hétérogènes, dynamiques et ouverts. Ils peuvent être composés de centaines de composants logiciels déployés sur des milliers d'appareils.

Le déploiement est un processus complexe qui a pour objectif la mise à disposition puis le maintien en condition opérationnelle d'un système logiciel. Pour les systèmes multi-échelles, l'expression du plan de déploiement ainsi que la réalisation et la gestion du déploiement sont des tâches humainement impossibles du fait de l'hétérogénéité, de la dynamique, du nombre, mais aussi parce que le domaine de déploiement n'est pas forcément connu à l'avance.

L'objectif de cette thèse est d'étudier et de proposer des solutions pour le déploiement de systèmes répartis multi-échelles. Nous proposons tout d'abord une mise à jour du vocabulaire relatif au déploiement, ainsi qu'un état de l'art sur le déploiement automatique des systèmes logiciels répartis. Le reste de la contribution réside dans la proposition : d'un processus complet pour le déploiement autonomique de systèmes multi-échelles ; d'un langage dédié (DSL), MuScADeL, qui simplifie la tâche du concepteur du déploiement et permet d'exprimer les propriétés de déploiement ainsi que des informations concernant la perception de l'état du domaine de déploiement ; d'un middleware, MuScADeM, qui assure la génération automatique d'un plan de déploiement en fonction de l'état du domaine, sa réalisation puis le maintien en condition opérationnelle du système.

Mots-clés

Déploiement - Autonomique - Systèmes multi-échelles - Composant - Systèmes répartis

---

Institut de Recherche en Informatique de Toulouse (IRIT), UMR 5505

Université Paul Sabatier, 118 Route de Narbonne, -31062 TOULOUSE CEDEX 9