Modélisation d'un intergiciel de grille pour le déploiement auto-adaptatif

Eddy Caron, Maurice Djibril Faye, Jonathan Rouzaud-Cornabas, Ousmane Thiaré

Modélisation d'un intergiciel de grille pour le déploiement auto-adaptatif

Eddy Caron*, Maurice Djibril Faye*†, Jonathan Rouzaud-Cornabas*
* LIP . Université de Lyon.
UMR CNRS - ENS Lyon - INRIA - UCBL 5668
Lyon, FRANCE
{eddy.caron, maurice.faye, jonathan.rouzaud-cornabas}@ens-lyon.fr

Ousmane Thiaré
† LANI
Université Gaston Berger
Saint Louis, Senegal
{ousmane.thiare}@ugb.edu.sn

***Abstract*—Les intergiciels pour grille-cloud sont des systèmes assez complexes. Cela rend leur exploitation très difficile pour les humains. En effet, en cas de défaillance ou de comportement non souhaité, le temps nécessaire pour localiser l'origine de la défaillance et pour prendre les mesures appropriées peut être trés long, laissant le système dans un état incohérent. Une solution pour remédier à cette limitation est de rendre ces systèmes auto-adaptatifs. Un tel système a la capacité de modifier en temps réel et de manière autonome (partiellement ou totalement) son comportement en réponse à des variations de son environnement. Nous partons d'un intergiciel pour grille et cloud et voulons rendre son déploiement auto-adaptatif. Pour avoir une idée du comportement des algorithmes auto-adaptatifs que nous avons conçus, nous avons choisi de les simuler. Pour ce faire, nous avons d'abord commencé par modéliser les différentes entités qui seront utiles pour la simulation, à savoir: l'infrastructure physique, l'intergiciel et le déploiement (l'ensemble constitué d'une instance de l'intergiciel en exécution sur des ressources physiques). Nous présentons dans cet article les modèles de ces trois entités.**

Keywords*—*Intergiciel; Déploiement; Modélisation; auto-adaptation; Grille, Cloud

I. Introduction

Les environnements distribués sont des systèmes complexes, généralement organisés (logiquement) en trois couches: une couche physique, une couche intermédiaire (appelée intergiciel) et une couche application. L'intergiciel offre une vue unique des ressources physiques en cachant aux applications et utilisateurs finaux l'hétérogénéité des ressources de la couche physique. Avant de bénéficier des avantages d'un intergiciel, il faut d'abord les déployer et ce déploiement doit satisfaire généralement un ensemble de contraintes. Des outils pour réaliser des déploiements, de la configuration existent [1]–[4]mais la plupart des déploiements réalisés sont statiques. Or, un déploiement statique sur un environnement dynamique comme les grilles ou les clouds n'est pas une bonne solution. Les variations de la plate-forme peuvent provenir de la défaillance de nœuds, de problèmes liés aux réseaux, du crash des processus de l'intergiciel. Elles peuvent aussi être inhérentes à la plate-forme comme dans le cas des clouds. Lorsque le déploiement est statique et qu'un comportement non souhaité est détecté, le seul moyen de réagir est souvent de reprendre tout le processus de déploiement, ce qui a un coût élevé.

Ainsi, faire de sorte de pouvoir prendre en compte les variations sans devoir tout reprendre à zéro présente des avantages certains.

Dans notre cas, nous voulons ajouter des fonctionnalités d'auto-adaptation à un intergiciel qui en était dépourvu pendant sa conception. Pour ce faire, nous allons introduire des algorithmes d'auto-adaptation. Pour simuler leurs comportements, nous allons réaliser un simulateur. Le simulateur a besoin d'un certain nombre de modèles. Il s'agit du modèle de l'infrastructure physique, de l'intergiciel (hiérarchique) et du modèle de déploiement. Dans cet article nous décrivons ces modèles. Les algorithmes d'auto-adaptation, leurs simulations et évaluations ne sont pas décrits ici.

La suite de l'article est organisée comme suit. La section II présente un exemple possible d'auto-adaptation d'un intergiciel comme motivation de notre travail. La section III présente sommairement quelques notions qui nous semblent utiles pour comprendre notre travail. La section IV introduit l'architecture que nous proposons pour le déploiement auto-adaptatif d'intergiciel. La section V présente les trois modèles requis: infrastructure, intergiciel et déploiement. La section VI présente quelques travaux connexes; elle est suivie d'une conclusion.

II. Motivation

La Figure 1 fournit un exemple de scénario qui met en avant la nécessité d'un processus de déploiement auto-adaptatif. β correspond au nombre de requêtes qui arrivent dans le composant d'ordonnancement de l'intergiciel. Cx/Sy correspond au cluster x sur le site y. Chaque cluster est composé d'un ensemble de nœuds de calcul Ni ($i \in N$). L'intergiciel est composé d'une hiérarchie d'instances de composants de base: MasterAgent (Mi), LocalAgent (Li), ServerDaemon (Si) ($i \in N$). La situation (1) décrit un ensemble d'instances de composant exécutées sur des nœuds et un ensemble de requêtes β en cours de traitement. En partant de cette situation, le nombre d'instances diminue si le nombre de requêtes décroît (élément 2) ou augmente si le nombre de requête croit (élément 3). C'est ce type de scénario (parmi d'autres) que nous voulons gérer grâce à notre architecture.

III. Contexte

A. Déploiement

Le processus de déploiement [5], [6] est une tâche complexe qui peut être subdivisée en deux phases. La première correspond à l'algorithme de planification qui alloue des ressources de l'infrastructure à des composants logiciels. La

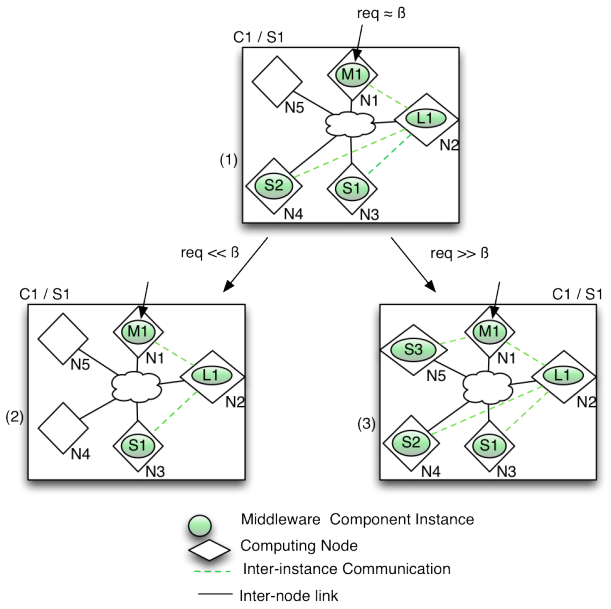

Fig. 1: Exemple d'un cas d'adaptation

seconde phase correspond aux actions de déploiement proprement dites (transfert de fichier, configuration des ressources, etc.).

B. Informatique autonome

L'informatique autonome [7] est une vision qui s'inspire des principes de l'automatique pour les appliquer dans le domaine du logiciel.

L'informatique autonome cherche à construire des systèmes auto-adaptatifs [8]–[10]. Un système auto-adaptatif peut, de manière autonome (totalement ou partiellement), détecter des variations de son environnement et réagir en cas de besoin. Généralement, la réalisation d'un système autonome se fait par l'implémentation d'une boucle de contrôle constituée de différents modules: surveillance, analyse des données, planification et exécution qu'on appelle structure MAPE-K [7]. Cette organisation est standard dans la plupart des systèmes autonomes. La Figure 2 donne une vue d'ensemble de l'architecture de notre système qui repose sur une boucle de contrôle. Les modules liés au processus de déploiement automatique sont colorés en bleu et ceux relatif aux aspects automatiques et de redéploiement sont en vert. On peut remarquer que cette division n'est pas stricte et que des modules peuvent relever des deux aspects.

C. Architecture de l'intergiciel DIET

Dans cette section, nous décrivons l'intergiciel qui nous sert de cas d'utilisation.

DIET [11] est un intergiciel GridRPC [12], qui est lui même une extension des Remote Procedure Call (RPC) appliquée au domaine des grilles de calcul. L'architecture par composant de DIET est structurée de manière hiérarchique pour améliorer le passage à l'échelle. DIET est implémenté en CORBA, et est constitué de plusieurs composants logiciel de base que sont: le client, le SED, les agents.Un **Client** est une application qui utilise l'infrastructure DIET pour résoudre un problème par l'utilisation de GridRPC. Un **SeD** (**Server Daemon**) joue le rôle de fournisseur de service. Le troisième composant de DIET, les **agents**, facilitent la localisation et l'invocation des services et donc l'interaction entre les clients et les SeDs. La hiérarchie des agents fournit des services de haut niveau comme l'ordonnancement et la gestion des données. Ces services permettent un passage à l'échelle grâce à leur distribution dans la hiérarchie des agents. Les agents sont de deux types: Les agents maître (**Master Agent ou MA**) et les agents locaux (**Local Agents ou LA**). Plusieurs hiérarchies peuvent être inter-connectées pour former une plate-forme multi-MA.

IV. ARCHITECTURE PROPOSÉE

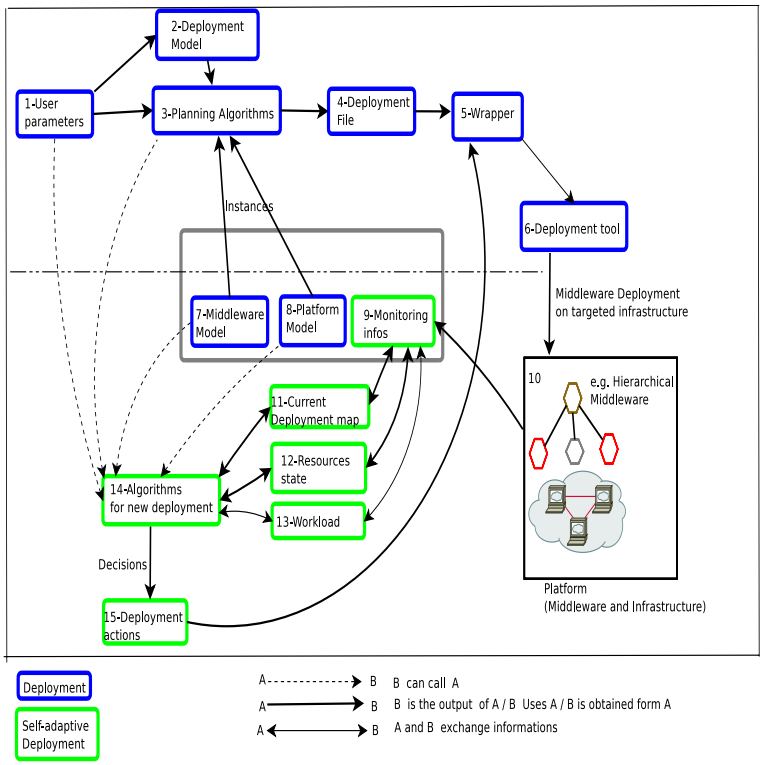

Fig. 2: Architecture pour le déploiement auto-adaptatif d'intergiciel

Dans cette section, nous décrivons l'architecture proposée. Nous commentons brièvement les différents modules en mettant l'accent sur les modules numérotés (2), (7), (8), comme indiqué dans la figure 2

Algorithmes de planification (3): la fonction d'un algorithme de planification consiste à répartir les composants de l'intergiciel sur les ressources de la plate-forme qui satisfont leurs besoins (si possible) et que certains objectifs prédéfinis par l'utilisateur soient atteints. Pour ce faire, les algorithmes ont besoin de connaître:

- **les paramètres de l'utilisateur (1)** qui expriment les objectifs que l'utilisateur souhaite atteindre.

- **le modèle de l'intergiciel (7)** qui décrit l'organisation des composants de l'intergiciel, les relations et les exigences (propriétés de la conception).
- **le modèle de plate-forme (8)** qui décrit les ressources et de leurs connexions.
- **la modélisation du déploiement (2)** n'est pas obligatoire, et peut parfois être exprimée par les paramètres de l'utilisateur. Il permet à l'utilisateur final de décrire une hiérarchie particulière de composants sans préciser où les composants seront placés (les algorithmes de planification vont le faire).
- **le fichier de déploiement (4)**: il s'agit de la sortie des algorithmes de planification. Il précise pour tous les composants de l'intergiciel les ressources qui leur sont allouées. Ce fichier a le même format que celui du module (2). Mais certains éléments qui ne sont pas pertinents peuvent être laissés vides. Il s'agit d'une instance du modèle de déploiement.

Le convertisseur (5): ce module convertit le fichier de déploiement (exprimé dans un format générique) en un fichier au format compris par l'outil de déploiement particulier utilisé (6). Il faut prendre de (4) les informations pertinentes et créer l'entrée de l'outil de déploiement (6) qui exécute les opérations de déploiement réelles [5], [6] comme le transfert de fichiers, la configuration des ressources ciblées, l'activation du processus, etc. Après les actions de (6), nous obtenons une hiérarchie de composants de l'intergiciel en cours d'exécution sur les ressources de l'infrastructure physique.

Information de surveillance (9): il surveille la plate-forme et recueille les informations. Ces informations concernent à la fois la plate-forme intergiciel et celle des ressources.

A partir des informations recueillies par (9), **la carte du déploiement courant (11)** est créée, c'est une mise en correspondance entre les composants intermédiaires à l'exécution et les moyens sur lesquels ils sont en cours d'exécution. Les états des ressources (12) sont obtenus et des mesures de la charge de travail de certains des composants intermédiaires sont prises. Ces trois modules peuvent communiquer avec l'outil de suivi et vice-versa.

Ces trois modules (11 à 13) sont des entrées des algorithmes de re-déploiement (14). Ces modules possèdent les connaissances nécessaires pour évaluer si un redéploiement est nécessaire. Si c'est le cas, le module regarde comment le faire en fonction de sa connaissance des paramètres actuels. Ce module s'appuie là-dessus et utilise la totalité ou une partie des algorithmes de planification (3). Les actions décidées (15) qui sont la sortie du module (14) sont dans le format d'un fichier de déploiement et peuvent avoir besoin d'être traduites par un convertisseur pour un outil de déploiement particulier qui permet d'exécuter les actions.

V. MODÈLES

Dans cette section, nous présentons les modèles correspondant aux modules (2), (7) et (8) de l'architecture proposée. Ces modèles ont été implémentés sous forme de schémas XML.

A. Modélisation des ressources de la plate-forme

La Figure 3 représente le modèle décrivant les ressources de l'infrastructure. Ce modèle permet une abstraction et simplification d'un système distribué réel tel qu'une grille, une fédération de clusters ou de clouds. Notre but est de fournir un modèle qui puisse représenter les différents types de systèmes distribués. Il se compose de plusieurs éléments:

Plateform: Cet élément représente la plate-forme (infrastructure). La plate-forme est composée d'un ensemble de ressources ainsi que d'une série de liens. Elle a un nom et une propriété "variabilité".). Cette propriété est définie par les administrateurs de la plate-forme. Son but est de capturer à quel point les paramètres considérés de la plate-forme sont variables (peu fréquente, fréquente, très fréquente). Cette valeur peut être calculée (ou estimée) en prenant en compte la variabilité de chaque ressource composant la plate-forme ou en analysant l'historique de la plate-forme. Dans le cas des stratégies auto-adaptative de redéploiement, il est utile d'avoir connaissance de la variabilité de la plate-forme. En effet, certaines stratégies de redéploiement peuvent être efficaces dans un milieu très dynamique et être moins performantes dans le cas de plate-formes peu dynamique.

Resource: les ressources sont de type: Cluster, Nœud de Calcul (**Node**) et Site. Une ressource a un identifiant et dispose de zéro ou plusieurs Localisations (élément **Location**). Chaque localisation est une pair (clef,valeur) qui spécifie un groupe auquel appartient la ressource. Chaque pair capture la notation d'appartenance/localisation d'une ressource. Par exemple, une ressource appartient à un site (site, nomSite), à un réseau local (reseauLocal, adresses IP) et à une ville (ville, nomVille). Certaines localisations peuvent être géographiques et d'autres liées au réseau. La localisation et l'appartenance à des groupes d'une ressources sont nécessaires pour certaines décisions de redéploiement quand il est nécessaire de déplacer un composant ou des données mais aussi pour des raisons de sécurité. Une ressource est aussi décrit par un ensemble de liens**Link**.

Link: Cet élément décrit les liens de communications entre les ressources (nœud, cluster, site). Il est rattachée à deux ressources (endpoint1 et endpoint2). L'attribut **linklevel** spécifie la nature et le niveau dans la hiérarchie du lien. **linklevel** peut avoir les valeurs suivantes: "intraCluster", "intraSite", "interSite", "interCluster" et "interNode". Le lien est aussi composé d'un ensemble de **Capacité** comme la bande passante et la latence.

Capacity : Une capacité est décrite par un nom (CPU, diskSpace, numberOfCore, etc.), une capacityUnit qui spécifie dans quelle unité la valeur de la capacité est exprimée, une capacityFlavor qui définit si la valeur de la capacité est représenté comme une valeur simple, un intervalle ou une liste et capacityValue qui contient la ou les valeurs.

Node : Un nœud est défini comme un ensemble d'éléments **Capacity**. Il a un identifiant qui est unique au sein du cluster auquel il appartient.

Cluster: Un Cluster est un ensemble de nœuds dont un (des nœud) a une fonction spéciale (appelé frontEnd). Un cluster virtuel tournant dans un cloud est représenté comme un cluster avec un ensemble de machines virtuelles (décrites comme des

nœuds) et un ensemble de liens de communications. Si un cluster appartient à un ou plusieurs sites, un ou plusieurs éléments **Location** sont utilisés.

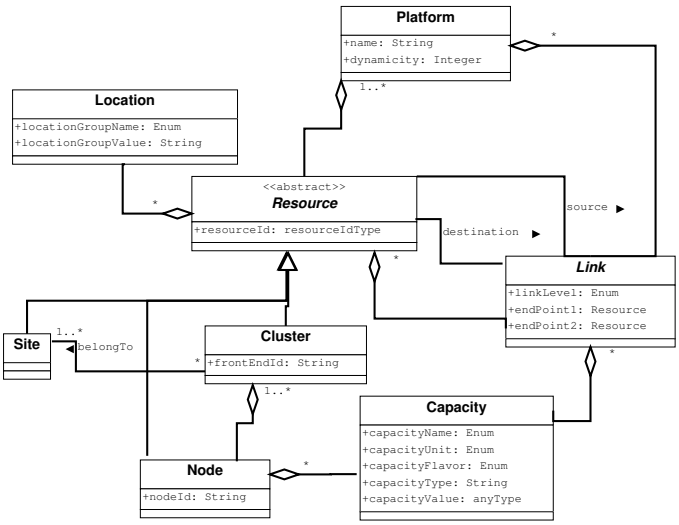

Fig. 3: Modèle d'infrastructure

B. Modélisation de l'intergiciel

La Figure 4 représente le modèle décrivant l'architecture de l'intergiciel (Middleware) qui nous sert de cas d'utilisation et d'autres qui ont la même structure. Il permet de construire une représentation abstraite et simplifiée d'un intergiciel hiérarchique, en particulier celui qui nous sert de cas d'utilisation pour tester les algorithmes de redéploiement. Notre but est de fournir un modèle qui permettra de représenter plusieurs types d'intergiciels hiérarchiques. Ce modèle est composée de plusieurs éléments:

Middleware: Il représente l'intergiciel et dispose d'attributs comme un nom et une version. Il est constitué d'un ensemble fini de composants de base. Ce sont des instances des composants de bases qui s'exécutent effectivement sur les machines physiques ou virtuelles.

MiddBaseComponent: Un composant de base de l'intergiciel contient un identifiant et contient les sections suivantes: **SoftwareInfo**: Cet élément décrit les informations logicielles du composant qui sont fixées durant la conception comme les binaires. **CommunicationInfo**: décrit pour un composant de base les autres composants avec qui les instances peuvent communiquer et comment est faite cette communication. La communication est décrite comme un ensemble d'exigences. Une exigence est exprimée comme une capacité (ce qu'elle exige doit faire partie des capacités d'une ressource). Un attribut spécifie si cette exigence est stricte (doit forcément être satisfaite) ou non (politique de "best effort"). **SoftwareRequirement**: Il représente les exigences logicielles d'un composant sur une ressource sur laquelle il peut être exécuté. Par exemple, un composant peut exiger un système d'exploitation spécifique. Il est composé d'un ensemble de capacités exigées. **HardwareRequirement**: Comme pour le SoftwareRequirement, cette section exprime un ensemble de capacités exigées mais celles-ci sont matérielles. Par exemple, le composant peut exiger un type particulier d'architecture processeur ou une quantité minimum d'espace disque. **LocalityRequirement**: Cette section décrit les exigences liées à la localisation du composant. Certains composant peuvent souhaiter partager la même localisation pour différentes raisons comme la rapidité des communications. **SecurityRequirement**: Cette section définit le niveau de sécurité qu'exige le composant. Cette valeur (basse, moyenne, haute) est interprétée par l'algorithme de déploiement. Elle est spécifique à chaque intergiciel.

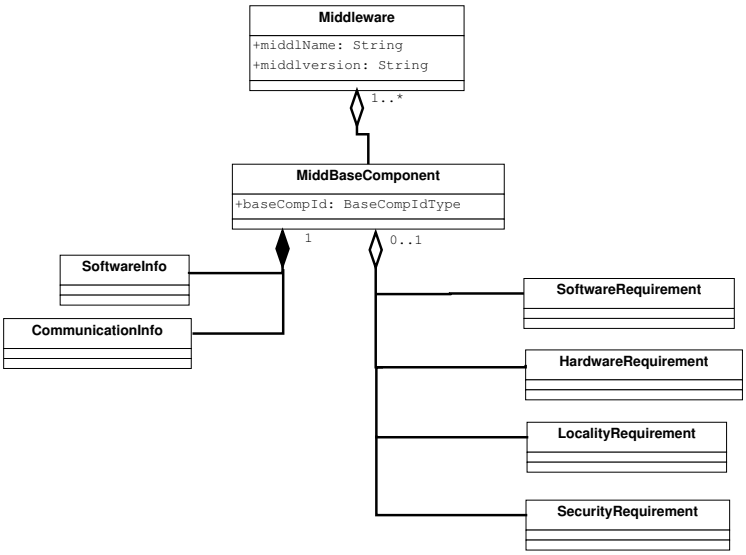

Fig. 4: modèle de Middleware

C. Modélisation d'un déploiement

Nous cherchons à capturer à travers ce modèle l'image d'une hiérarchie de composants de l'intergiciel en exécution sur des ressources physiques. En d'autres termes, on cherche à modéliser la répartition des composants de l'intergiciel en même temps que les ressources physiques(Cluster, Nœud,...) sur lesquelles les composants s'exécutent.

La Figure 5 représente le modèle d'un déploiement. Un déploiement est décrit par les éléments suivants et leurs relations :

Hierarchy: La Hiérarchie est un ensemble d'instances (**DeployedInstance**) de composants de base et de leurs liens de communication. Chaque instance contient les informations et exigences nécessaire pour l'algorithme de planification et pour l'outil de déploiement.

DeployInstanceInfo: Cet élément contient les informations relatives à l'instance. Il contient un ensemble d'exigences présentées ci-dessous:

LocalityRequirement: Il spécifie qu'une instance doit appartenir à une Localisation (réseau ou géographique). Cette exigence peut être stricte ou flexible. **ColocationRequirement**: Il spécifie si l'instance doit avoir la même localisation que d'autres instances (par exemple même site, même cluster, même Nœud ,même réseau local, ...). **SecurityRequirement**: Cet élément est utilisé pour spécifier l'exigence de sécurité de

l'instance du composant. **LinkRequirement**: Chaque instance déployée a un ensemble d'élément **LinkTo** qui spécifie les exigences de communication: **LinkTo**: Il décrit qu'une instance de composant est connecté à une autre instance. Dans le cas d'un intergiciel hiérarchique, une instance est au moins connecté à une autre. Cet élément peut aussi contenir un ensemble d'exigences sur le lien de communication qui guidera l'algorithme de planification.

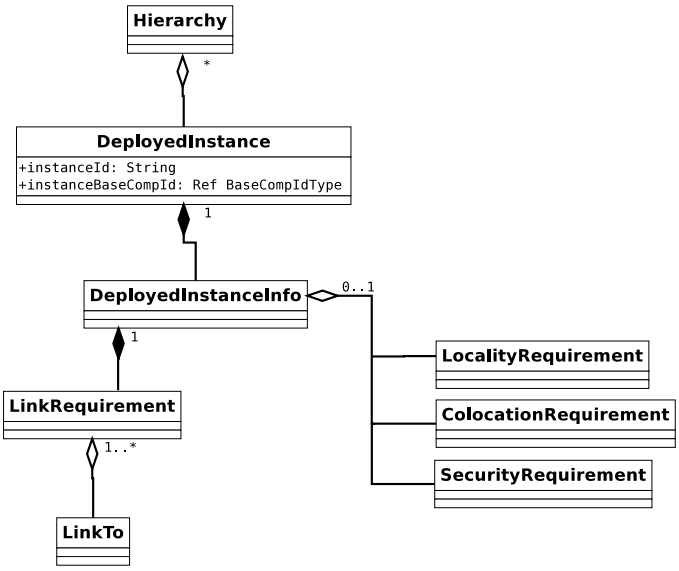

Fig. 5: Modèle de Déploiement

VI. Travaux Connexes

[13] propose un langage de description de ressources et leurs interconnections. L'accent est mis sur la description d'un réseau de machines virtuelles. [14] permet une description assez fine de la structure des ressources physiques de type cluster et offre une API permettant de lancer des requêtes pour obtenir des informations sur la structure de la ressource entre autres. [15] présente un modèle pour les intergiciels hiérarchiques et des algorithmes pour déployer une hiérarchie d'ordonnanceurs sur un cluster ou une grille. Cependant, une sévère limitation de ce travail est qu'un seul type de service peut être déployé dans la hiérarchie. [16] propose une solution à cette limitation et présente un modèle et un algorithme pour le déploiement de intergiciel hiérarchique sur une plate-forme homogène mais ne s'intéresse pas à l'aspect auto-adaptatif. Dans [17], les auteurs présentent un modèle étendu pour le problème du placement de composants. [18]présente une boite à outils pour le déploiement auto-adaptatif de système orienté service.

VII. Conclusion

Notre objectif est de réaliser un déploiement auto-adaptatif d'un intergiciel. Cette tâche est divisée en deux parties: la conception de modèles pour décrire l'infrastructure, l' intergiciel ainsi que le déploiement; la conception d'algorithmes de (re)déploiement et de planification. Ces algorithmes utilisent les connaissances fournies par les modèles et surveillent le déploiement afin de prendre des décisions qui permettront au système de s'auto-adapter. Dans cet article, nous avons présenté les trois modèles que nous avons utilisé pour simuler des algorithmes d'auto-adaptation pour un intergiciel de grille et Cloud.

References

[1] E. Caron, P. Chouhan, and H. Dail, "GoDIET: A Deployment Tool for Distributed Middleware on Grid'5000," in *EXPGRID workshop. Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Apllications and Tools. In conjunction with HPDC-15*, IEEE, Ed., Paris, France, June 19th 2006, pp. 1–8.

[2] Z. Hou, J. Tie, X. Zhou, I. Foster, and M. Wilde, "ADEM: Automating Deployment and Management of Application Software on the Open Science Grid," in *Grid 2009, 10th IEEE/ACM International Conference on Grid Computing*, IEEE. Banff: IEEE, October 13-15 2009, pp. 130–137.

[3] Chef, "https://www.chef.io/chef/," 2015.

[4] Puppet, "https://puppetlabs.com/," 2015.

[5] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. V. Der, and E. L. Wolf, "A Characterization Framework for Software Deployment Technologies," Department of Computer Science, University of Colorado, Tech. Rep., Apr. 10 1998.

[6] *Deployment and Configuration of Component-based Distributed Applications Specification*, Object Management Group, Inc., 2006, an Adopted Specification of the Object Management Group, Inc.

[7] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[8] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[9] J. Andersson, R. De Lemos, S. Malek, and D. Weyns, "Modeling dimensions of self-adaptive software systems," *Software Engineering for Self-Adaptive Systems*, pp. 27–47, 2009.

[10] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems," *Computing*, vol. 91, no. 1, pp. 43–73, 2011.

[11] E. Caron and F. Desprez, "DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid," *International Journal of High Performance Computing Applications*, vol. 20, no. 3, pp. 335–352, 2006.

[12] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, "Overview of GridRPC: A Remote Procedure Call API for Grid Computing," in *Grid Computing - GRID 2002, Third International Workshop*, ser. LNCS, M. Parashar, Ed., vol. 2536. Baltimore, MD, USA,: Springer, Nov. 2002, pp. 274–278.

[13] G. P. Koslovski, P. V.-B. Primet, and A. S. Charao, "Vxdl: Virtual resources and interconnection networks description language," in *Networks for Grid Applications*. Springer, 2009, pp. 138–154.

[14] B. Goglin, "Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc)," in *High Performance Computing & Simulation (HPCS), 2014 International Conference on*. IEEE, 2014, pp. 74–81.

[15] P. K. Chouhan, "Automatic Deployment for Application Service Provider Environments," Ph.D. dissertation, Ecole Normale Supérieure de Lyon, 2006.

[16] B. Depardon, "Contribution to the Deployment of a Distributed and Hierarchical Middleware Applied to Cosmological Simulations," Thesis, École Normale Supérieure de Lyon, october 2010.

[17] T. Kichkaylo and V. Karamcheti, "Optimal resource aware deployment planning for component based distributed applications," in *The 13th High Performance Distributed Computing*, June 2004.

[18] S. van der Burg and E. Dolstra, "Disnix: A toolset for distributed deployment," *Science of Computer Programming*, 2012.