Xj-ASD: Towards a j-ASD DSL eXtension for Application Deployment in Cloud-Based Environment

Kanga Koffi[1(✉)], Babri Michel[2], Brou Konan Marcelin[2], and Goore Bi Tra[2]

[1] Ecole Doctorale Polytechnique de l'Institut Nationale Polytechnique Félix Houphouët Boigny (EDP/INPHB), Côte D'ivoire UMRI 78: Electronique et Electricité Appliquée Laboratoire de recherche en informatique et télécommunication, Yamoussoukro, Côte d'Ivoire
koffi.kanga@larit.net

[2] Institut Nationale Polytechnique Félix Houphouët Boigny (INPHB), Côte D'ivoire UMRI 78: Electronique et Electricité Appliquée, Laboratoire de Recherche en Informatique et Télécommunication, Yamoussoukro, Côte d'Ivoire
{michel.babri,kmbrou,goore}@inphb.edu.ci

Abstract. In this paper, we propose an extension of the grammar of an application deployment constraints description language from a cloud computing platform. To do this, we draw a meta data model proposed by [1] for an application deployment in a cloud. This meta model, we extend j-ASD for the consideration of compatibility constraints or conformity between the virtual image data file formats used by the components of the virtual machines and those target sites that are deploying Smartphones, PC, etc. …

Indeed for a full deployment of applications from a cloud environment on high mobility rate (Smartphone, PC, etc. …), it occurs to ensure compliance of the data formats of these environments to that of Cloud platform. This conformity proves a prerequisite for deployment on a device from a cloud. To address these compliance constraints, we formalize them in matrix form and propose the use of a constraint solver.

Keywords: Application deployment · Software component description language constraint · Cloud computing · Deployment plan

1 Introduction

Designing application requires an approach called life cycle. This approach, in whatever form (V, waterfall, spiral …) includes a number of activities (design, implementation, validation, deployment, and administration) regardless of the approach [1]. These activities include the deployment is a complex process ranging from the production of the application and uninstall it [2].

T.F. Bissyande and O. Sie (Eds.): AFRICOMM 2016, LNICST 208, pp. 122–132, 2018.
https://doi.org/10.1007/978-3-319-66742-3_12

Today, with the emergence of the Internet of Things monitoring the development of service models in the Cloud, mobile device users want to use their applications on their phone, tablet and other materials with high rates of mobility. In this context, the deployment of applications becomes an important activity with its constraints corollary, given the diversity of deployment sites and components that make up these applications.

Faced with this dilemma, the research tried to find solutions to the architecture and deployment platform [2], the definition of deployment constraints languages [3] to describe the application to deploy, and the constraints of facilities. In this paper we propose an extension of this language to the specificities of a Cloud to benefit users of the benefits in terms of computing power and the Cloud profitability. Also in order to cover all the deployment activities, we integrate other deployment constraints are not supported, namely the management of data formats, image management from various file virtualization to deploy. The rest of the paper is organized as follows. In Sect. 2, we are a state of the art in this field. Section 3 is devoted to our contribution. We end with a conclusion while generating few prospects for our future work.

2 State of the Art

Several research studies describing the tools and procedures around the deployment exist. But to our knowledge, these are almost always intended for fixed topologies machines and/or known at the time of deployment and therefore not relevant to our context. This section presents some research related to the application deployment.

Fractal Deployment Framework (FDF) [4], is a tool that provides a generic in deploying applications. It consists of a deployment description language, a set of components, and user interfaces.

The deployment unit is an archive that contains the binaries and software deployment descriptor. The main limitation of this tool is the static nature of the deployment although a static deployment plan qualifies in relatively stable environments such as grid computing, this type of tool cannot be used in environments characterized by a topology network dynamics as cloud environments. Another limitation of FDF is that it does not provide heuristic dynamic reconfiguration that allows the incorporation of machinery malfunction situations for example.

Software Dock [5], it provides a Framework for the configuration and deployment of software. It uses a system of events and mobile agents to control deployment activities such as installation and activation. Deployment life cycle includes the installation, activation, deactivation, updating, uninstalling and reconfiguring. The deployment system uses a client/server architecture associated with event management system. A server called “release dock” is installed at the manufacturer’s website. A customer called “fi eld dock” is installed at each site software consumers, which acts as an interface for the release dock. However, Software Dock does not allow the description of the software

architecture and deployment constraints. Software Dock also offers a centralized, static deployment process that does not meet the needs of dynamic reconfiguration and deployment of open environments.

R-OSGi [6] is a middleware that uses the standards of the OSGi specification to support the management of distributed modules. Upon deployment, R-OSGi can be used to execute a distributed application simply indicate the deployment locations of deferent modules. The developer of an R-OSGi application has full control over how the application is distributed. Manual control of the deployment process and its configuration in a large scale environment left is a very complex task and represents for us a very important human intervention in the deployment process. In addition, R-OSGi is only intended to create static software deployments that cannot be used in environments distributed large-scale open as ubiquitous systems and P2P.

Dearle et al. proposed in **[7]** middleware, MADME (Monitoring Automatic Deployment and Management Engine) for deploying and managing applications consist of one or more components called Cingal-bundle. Deployments constraints are specified with the Deladas language. The deployment administrator specifies an initial deployment target, then the deployment system tries to generate a configuration that describes the process of deploying the application components. After initial deployment, the deployment system verifies the satisfaction of the initial target and redeploys the application if necessary. This approach has similar motivations to ours. Indeed, one of the reasons is the reduction of human intervention in the deployment process by automatically generating the deployment plan. However, the proposed middleware is not usable in environments with an unpredictable topology. In addition the user of the tool must restart the entire deployment process. Upon the occurrence of a disconnection or failure for example.

A DSL-Based Approach to Software Development and Deployment on Cloud [8]: In this work, Krzysztof Sledziewski et al. present an approach incorporating a DSL for the development and deployment of applications in a Cloud. In this approach the authors propose that developers use a DSL for describing the model associated with the application. This model is then translated into a specific code and automatically deployed in a cloud. This approach is specific to a deployed in a Cloud and facilitates the work of the administrator to deploy. However, the proposed approach does not take into account the conditions or deployment constraint to satisfy.

A DSL for Multi-scale Deployment and Autonomic Software [9]: In this article, Raja BOUBEL and Al. present a progressive work that aims to define and test a DSL dedicated to autonomic application deployment in multi-scale environments. In these environments, the network topology may vary according to hardware failures.

In this work the authors design a DSL to support the expression of constraints and properties related to autonomic application deployment in multi-scale environments. However, they do not provide in their DSL prior restraint activation, deactivation, installation, application uninstall.

J-ASD [3]: A middleware for autonomic software deployment. It consists of a set of software that can best meet the deployment problem of a distributed application regardless of the execution context. In other words this middleware performs independently (with minimal human intervention as possible) a deployment that meets a set of constraints defined by the deployment administrator. It is able to self- adapt and automatically resolve some problems associated with the instability and the opening of the environment. It is based on:

- A specific language (DSL) for the description of deployment constraints called j- ASD DSL
- A network service to automatically detect deployment target sites
- A bootstrap middleware for the preparation of the execution environment
- A constraint solver for solving constraints and deployment plan generation
- A deployment support and an adaptable mobile agent system for the execution and supervision of deployment activities
- A deployment algorithm.

However j-ASD, in defining the conditions to be met for a deployment does not take into account the level of use of the battery devices, characteristics that are related to cloud virtual machines and images of virtualized applications, formats data images to deploy.

3 Contribution

In view of the existing work, more deployment-related issues seem to be resolved. In particular, the deployment constraints related problems (install/uninstall). Nevertheless, some aspects of the deployment as part of activation, deactivation and update components seem not yet to find solutions.

An analysis of existing studies shows that configure and deploy of application in a large scale environment such as the Cloud is not easy. This complexity is due to the multitude of components, the heterogeneity and the large number of target sites in an environment with high levels of mobility (and therefore variable topology) that make up an application. So our contribution is as follows:

- Taking into account the given file formats to deploy the devices from virtual machines in cloud platforms.
- Extension of the grammar description language deployment constraints J-ASD DSL initiated by [3] to take into account other constraints (i) pre-deployment, (ii) relating to the use of the battery deployment target sites from a cloud platform (iii) network latency which is based on the deployment plan and also the power of the processors.

3.1 Modeling the Inclusion of Data Formats (Fig. 1)

In Fig. 1, our model shows different class with roles based on their attributes.

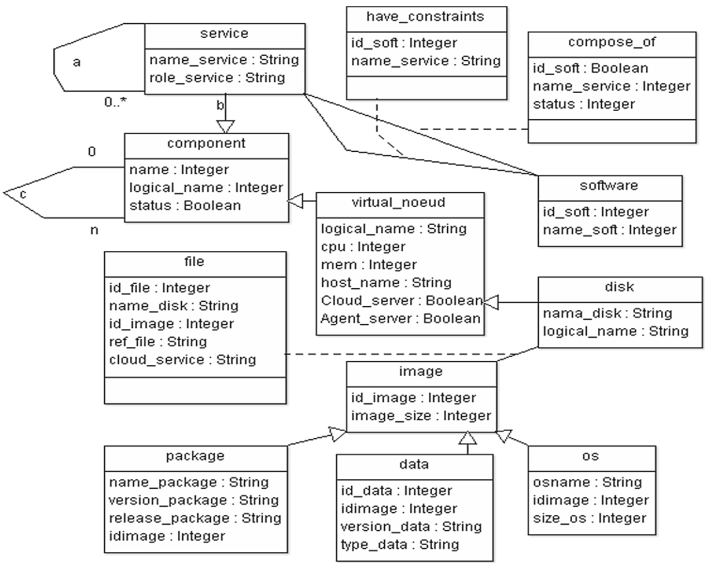

Fig. 1. Meta data model for deployment in the cloud [1]

In the following table, we present these different classes and their role in the process of deploying an application (Table 1).

Table 1. List of class in Fig. 1 and their respective roles

Class	Roles
Application	Is characterized by an application name, version, and an ID. It may consist of components (class has) and also its use requires the satisfaction of certain vis-à-vis stress-related services to its components (stress class)
Composant	A component appears as the component of an application. It is also characterized by a name, version and status (on/off)
Service	This class is characterized by name, a role and also may depend on the operating state of other services (reflexive link a)
Contrainte	Determines if a service is available for the proper functioning of an application. In which case an adaptation of the application to another service arose

(*continued*)

Table 1. (*continued*)

Class	Roles
Comporte	Materializes the creation of an application from components. For proper operation, the component needs the availability of several services (b) or as other components (reflexive link c)
Virtual_noeud	Represents a virtual machine. It is characterized by a name, CPU frequency, memory value, 2 Boolean attributes (cloudserver and agent_cloud) whether the node is a server or client
Disk	This class embodies the place of deployment of the virtual machine. It is characterized by a logical name (LOGICAL_NAME), size (size), an identifier (name_disk). It consists of a set of files (file).
File	Represents the element to be virtualized to form an image. It has the following features: a Id_image representing the corresponding image, the disk on which it is stored, its reference (path, a boolean flag to see if it is a file server to a virtual machine server
Image	Represents the file element (file) virtualized. it is characterized by the set of packages that make up the data and the OS with which it is compatible
Package	Each package of the image has a name (name_package), a version (version_package) and a release (release_package) that distinguish it from other packages
Data	Here this class characterizes the image because it determines the version (version_data) Image data type (or format) (type_data) data, and the data source (source_data)
OS	This class in turn determines the operating system on which an image can be deployed. It has a name (osname), size (size) for the space installation requires a distribution (os_distribution) and version (os_version)

3.2 Constraints Description Language Extension j-ASD DSL Based on the Model of Fig. 1

J-ASD DSL is a language with a simplified and intuitive grammar. This grammar is developed using Xtext[1] which is a specific languages Development Framework. As stipulated in the j-ASD DSL precursors, an application consists of one or more components. Each is defined by an identifier, a version, execution url or implementation but also by a set of software dependencies, hardware constraints and deployment constraints. So from the j-ASD DSL grammar presented in a we extend this grammar to battery usage constraints (PowerPref) potential deployment sites, internet network latency constraints (NetLatency) and also to the constraints CPU power (MIPSPref Million Instructions Per Second) and also to the constraints of the data format as shown in b in the same conditions given the high rate of mobility in Cloud environments.

[1] http://www.eclipse.org/Xtext.

a- J-ASD DSL language grammar Xtext

```
grammar eu.itsudparis.inf.JASDDsl with
org.eclipse.xtext.common.Terminals
generate jASDDsl
Model:
Software=Software
Components+=Component+(HostConstraints+=HostConstrain
t*)?
Deployment=Deployment;
Software:
"Software" "{"
"Name" "=" name=ID
"Version" "=" ver=INT
"Components" "=" components+=(ID)*
"}";
Component:
"Component" "{"
"Name" "=" name=ID
"Version" "=" ver=INT
"Url" "=" url=STRING
("Dependencies" "=" dependencies+=ID*)?
"}";
HostConstraint:
"HostConstraint" "{"
"Name" "=" name=ID
constraints+=(OsPref | CPUPref |
RAMPref | HDPref | NetSpeedPref)*
"}";
Deployment:
"Deployment" "{"
{Deployment} members+=MemberDecl*
"}";
MemberDecl:
component=ID "@" localisation=Localisation
("with" constraints+=(ID)*)?;
OsPref:
"OSNameContains" name=STRING;
CPUPref:
"CPULoad" InfSup val=INT "%";
terminal InfSup:
"<" | ">" | ">=" | "<=";
RAMPref:
"RAM" sym=InfSup val=INT "MB";
HDPref:
"HD" sym=InfSup val=INT "MB";
NetSpeedPref:
"NetSpeed" sym=InfSup val=INT "kb/s";
Localisation:
IPv4 | NetName | Val | Interval | All;
terminal IPv4:
INT '.' INT '.' INT '.' INT;
NetName:
STRING;
Val:
INT;
terminal Interval:
INT ".." INT;
terminal All:
"all";
```

b- Extension given to grammar j-ASD DSL language

```
‘contribution to the extension of the grammar
'J-ASD DSL# puissance de la batterie
PowerPref :
"Power" sym=InfSup Val=INT "%";
# network latency
NetLatency :
"NetLat" sym=InfSup Val=INT "ms";
# puissance du processeur
MIPSPref :
"MIPSPval" sym=InfSup Val=INT "MIPS";
‘Extension into account the meta data model
# Data format
Format :
"format" "=" name=STRING ;
DataConstraint:
"dataConstraint" "{"
"Name" "=" name=ID
"Version" "=" ver=INT
"type" "=" name=STRING
"source" "=" name=STRING
"}";
```

So as defines our extension grammar highlights a number of constraints to be satisfied to make a full deployment. In this work, as we make a deployment from a cloud, we propose the use XMPP protocol [11] for the management of network discovery sites belonging to the deployment plan. If network discovery services are defined and the constraints of compliance (yes compatibility) of defined data format property, our second contribution is working to formalize and solving these constraints as a constraint satisfaction problem (CSP) that can find solution using a constraint solver. As part of our prototype we chose the open source constraint solver Choco [11] to be consistent with [9].

3.3 Formalization and Resolution of Constraints

As part of j DSL-ADS, the constraint satisfaction problem is constructed from a set of integer variables (compliance matrix) and a set of constraints on these variables. Under these conditions we model the CSP program with the following:

- A set C software components forming the application to deploy
- Let $C = \{C1, C2, C3, \ldots, Cn\}$
- A set S deployment target sites detection network discovery service
- A given compliance matrix (Cfm) modeling the compliance or non-compliance of file images of the data formats supported by virtual machines from components that they have with those of deployment sites is such that:
  - Cfm (Ci, Sj) = 1, if the component Ci has the same data format as the site Sj
  - Cfm (Ci, Sj) = 0, if the component Ci has the same data format as the site Sj
- A Q set of constraints on the Si sites (e.g. Powerload, Netlatency …)
- A set of constraints on the variables Cfm (Ci, Sj)

c-Example of J-ASD Program DSL written taking into account the constraints of data format compliance for deployment

```
Software {
Name=niveau_de_test
Version=1
Components=ramSize display
}
Component {Name=RamSize
Version=1
Url=http://x.fr/RAM-Size.jar }
Component {Name=display
Version=1
Url="http://x.fr/Display.jar"}
HostConstraint {Name=Display-Constraint
CPULoad > 80%
RAM >= 40 MB
OSNameContrains "Linux"}
Deployment {
RamSize @ all
display @ 2 with Display-Constraint
}
**********
extension taking into account our contribution to the constrained
**********
*** definition of the characteristics of the component niveau_batterie
Component {
Name=niveau_batterie
Version=1
Url="http://x.fr/niveau_batterie.jar"
}
*** definition of CONSTRAINT deployment contraint3
dataConstraint {
Name=contraint3
version = 2
Type = ".exe"
OSNameContains "Windows"
}
*** component deploying niveau_batterie
*** on all sites is considering the contrainte3
Deployment {
niveau_batterie @ all with Constraint3
}
```

In this j-ASD DSL program, we have a description of deploying an application called "niveau_de_test" consists of the following components: ramsize - Display - niveau_batterie characterized by their name (Name), their version, the URL of storage. The program also includes a set of constraints (Display-constrained constraint3), which are constraints on the size of the RAM memory, the processor occupancy and operating system.

The deployment constraints (activation) Niveau_batterie mean that the component must be deployed on all sites that respect Contrainte3 constraint. This constraint on the data format of the images of virtual machine files.

Formally this means:
Niveaubatterie ∀ ∈ C, ∀ If ∈S
If ((version = 2) and (type = ".exe") and (osname = "Windows")) then
Cfm (niveaubatterie, Si) = 1
else Cfm (niveaubatterie, Si) = 0

As for the second constraint, it means that the display component must be deployed (on) a set of two sites that satisfy the constraint "Display-Constraint", it is formally expressed by:

Display ∀ ∈ C, ∃ S1, S2 ∈S such que:
If ((CPULoad> 80%) and (ramsize ≥ 40 MB) and (osname = "linux")) then
Cfm (display, S1) = Cfm (display, S2) = 1
else
Cfm (display, S1) = Cfm (display, S2) = 0

4 Conclusion

In this article we presented our contribution to the application deployment problem solving from a Cloud by providing an extension to the grammar of deployment constraints description language j-ASD DSL. These deployment constraints relate to the compatibility of data formats virtual images of the component files. This set of constraints is in solution using a constraint solver for calculating a deployment plan.

Our proposed extension can be used to manage the power consumption, latency network's management to ensure full deployment from a cloud-based environment. Also it helps enable deployment of equipment in a variable topology environment.

For now we continue our work with the deployment of OSGi Framework. However, the use of new deployment unit as SCA applications (Service Component Architecture) (Open Service Architecture Collaboration 2007) with Frascati platform [Seinturier 2009 Seinturier 2012] as deployment media is a natural extension of j-ASD DSL.

References

1. Etchevers, X.: Déploiement d'applications patrimoniales en environnements de type informatique dans le nuage. Other. Université de Grenoble, 2012. French. <NNT: 2012GRENM100>. <tel-00875568>
2. Dibo, M.: UDeploy: une infrastructure de déploiement pour les applications à base de composants logiciels distribués. Other. Université de Grenoble, 2011. French. <NNT: 2011GRENM001>. <tel-00685853>

3. Matougui, M.E.A., Leriche, S.: j-ASD: un middleware pour le déploiement logiciel autonomique. NOTERE/CFIP'12: Conférence Internationale Nouvelles Technologies de la Répartition/Colloque Francophone sur l'Ingénierie des Protocoles, Oct. 2012, Anglet, France. Cepadues. <hal-00757154>
4. Quinton, C., Duchien, L.: Vers un Outil de Configuration et de Déploiement pour les Nuages. JLdP - Journee Lignes de Produits, Nov 2012, Lille, France. pp. 83–94. <hal-00747319>
5. Flissi, A., Dubus, J., Dolet, N., Merle, P.: Deploying on the grid with deployware. In: CCGRID, pp. 177–184 (2008)
6. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: Cook, W.R., Clarke, S., Rinard, M.C. (eds.) Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, Reno/Tahoe, Nevada, USA. SPLASH/OOPSLA Companion, pp. 307–309. ACM, October 2010. doi:10.1145/1869542.1869625
7. Rellermeyer, Jan S., Alonso, G., Roscoe, T.: R-OSGi: distributed applications through software modularization. In: Cerqueira, R., Campbell, Roy H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 1–20. Springer, Heidelberg (2007). doi:10.1007/978-3-540-76778-7_1
8. Dearle, A., Kirby, G.N.C., McCarthy, A.: A framework for constraint- based deployment and autonomic management of distributed applications. CoRR, vol. abs/1006.4572 (2010)
9. Sledziewski, K., Bordbar, B., Anane, R.: A DSL-based approach to software development and deployment on cloud. In: 24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia. AINA, pp. 414–421. IEEE Computer Society, April 2010. doi:10.1109/AINA.2010.81
10. Boujbel, R., et al.: A DSL for multi-scale and autonomic software deployment. In: The Eighth International Conference on Software Engineering Advances, ICSEA 2013, pp. 291–296 (2013)
11. The Choco Team: Choco: an open source java constraint programming library. Ecole des Mines de Nantes, Research report (2010). http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf
12. Saint-Andre, P., Smith, K., Tronçon, R.: XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies. O'Reilly Media, Inc. (2009)