MuScADeL: A Deployment DSL based on a Multiscale Characterization Framework

Raja Boujbel, Sam Rottenberg, Sébastien Leriche, Chantal Taconet,
Jean-Paul Arcangeli, Claire Lecocq

MuScADeL: A Deployment DSL based on a Multiscale Characterization Framework

Raja BOUJBEL*, Sam ROTTENBERG†, Sébastien LERICHE‡,
Chantal TACONET†, Jean-Paul ARCANGELI* and Claire LECOCQ†
* Université de Toulouse UPS, IRIT CNRS UMR 5505 France *firstname.lastname@irit.fr*
† Institut Mines Télécom/Télécom SudParis, CNRS UMR 5157 SAMOVAR France *firsname.lastname@telecom-sudparis.eu*
‡ Université de Toulouse, ENAC France *firstname.lastname@enac.fr*

***Abstract*—With the Internet of Things (IoT) paradigm, ambient systems move from locally distributed systems to Internet distributed systems. These systems become huge in term of number of devices and imply high heterogeneity (e.g., of devices, of networks). They are continuously evolving with appearing and disappearing devices at runtime. The inner complexity of these systems, called multiscale systems, requires autonomic deployment middleware. Such middleware should deploy components where and when necessary, and adapt the architecture of the deployed systems considering the different scales of the systems. In this paper, we define MuScADeL, a domain-specific language (DSL) dedicated to multiscale and autonomic software deployment. MuScADeL allows designers to abstractly define deployment properties without exact knowledge of the devices and networks the system will be deployed on. This DSL is based on a scale-awareness framework, which helps designers to characterize the multiscale nature of a system from several viewpoints such as device, network, administration and geography. With MuScADeL, deployment designers may express multiscale properties of systems to deploy. MuScADeL is a building block for deployment middleware that targets multiscale distributed systems. We illustrate the possibilities of MuScADeL through a smart transport scenario.**

***Keywords*-Multiscale distributed systems, Software deployment, Domain-specific language**

I. Introduction

Smart systems built over the IoT are distributed on many various computing nodes from smart sensors and mobile computers to cloud computers. Each family of host devices deals with its specific set of technologies, languages and communication protocols. Systems cross geographic and administrative boundaries, and each geographical or administrative domain requires specific rules (e.g., of communications, of privacy). We call those systems *"multiscale systems"*. A major issue with multiscale systems is to deal with high levels of heterogeneity.

IoT projects such as SmartSantander [1] and IoTI [2], come with examples of smart applications deployed at a city scale. Those applications are composed of a mass of software components in interaction. The distributed sets of machines that host them (called *deployment domain*) are huge, and present high heterogeneity (e.g., of devices, of networks). Moreover, these systems are mostly pervasive, and highly dynamic with devices that appear and disappear during the life of the system. Despite this, IoT projects and ambient projects rarely address software deployment issue. Nonetheless, we believe that smart deployment middleware that autonomically deploys multiscale systems, becomes essential. These middleware should adapt the architecture of the deployed systems considering the different scales of the systems by deploying components where and when necessary.

In this paper we present MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*), a domain-specific language (DSL) dedicated to autonomic software deployment of multiscale systems. MuScADeL allows designers to declare multiscale deployment properties without exact knowledge of the deployment domain. For example, the designer may specify that one server should be deployed in each local area network, or in each district of a city. One foundation of MuScADeL is MuScA (***Mu**ltiscale distributed systems **Sc**ale **A**wareness framework*), a scale-awareness framework, which includes a multiscale characterization process, and multiscale probe generation software. Through a model driven approach, MuScA helps designers to identify the multiscale nature of each system from its relevant viewpoints: for example device, network, geography or administration viewpoints. Besides, we show how multiscale probes are produced from MuScA models and how they are referenced by MuScADeL. Those probes provide scale-aware capability to future deployment middleware. This paper focuses on the MuScADeL DSL and on the multiscale probes, others parts of deployment middleware are not presented.

The organization of the paper is the following. In Section II, we motivate our work through a smart transport scenario. We depict our approach in Section III. Then, in Section IV, we position our work compared to the main DSL for software deployment and their ability to handle multiscale concerns. In Sections V and VI, we respectively present the MuScA characterization framework and the MuScADeL DSL. We show how to express multiscale deployment properties for the transport scenario. Finally, we conclude in Section VII.

II. Multiscale deployment motivations

In this section we introduce our deployment and multiscale vocabulary, and present a scenario that will support our illustrations through the paper.

A. Deployment and multiscale concepts

Software deployment is a post-production process that consists in making software available for use and then keeping it operational. It is a complex process that includes a number of inter-related activities such as installation of the software into its environment (transfer and configuration), activation, update, reconfiguration, deactivation and deinstallation [3].

A *deployment plan* is a mapping between a software system (a system of software components) and the deployment domain, completed with configuration data. It must take into account two kinds of dependencies: (1) between the components themselves and (2) between the components and their runtime environment. At runtime, the software system is deployed on the hosts in accordance with the deployment plan. Traditionally, the mapping and the deployment activities are undertaken or controlled by a human operator.

Multiscale systems have very complex architectures. In order to deploy these systems, and more specifically to express the properties, it is necessary to be able to describe precisely their architecture. Therefore we propose a multiscale vocabulary that aims at describing the architecture of multiscale distributed systems. As presented in [4], the architecture of a system is obtained from the study of this system from different *viewpoints*, each viewpoint leads to a *view* of the system. Following this general approach, we study multiscale distributed systems from different viewpoints. There are many viewpoints and views to consider: e.g., devices included in the system, networks crossed for interacting, administrative or social organization of the users of the system, geographical distribution of the system. Then, each viewpoint of a multiscale system can be studied through several analysis *dimensions*, associated with *measures*, to define the different *scales* of the system. For example, the device viewpoint can be analyzed through the storage capacity dimension, measured in bytes, which leads to identify different scales of devices in terms of storage capacity: e.g., kilobytes scale, gigabytes scale and petabytes scale. This multiscale vocabulary is more detailed and formalized in Section V-B.

B. Motivating scenario

In this section, we present a motivating scenario inspired from smart city projects such as CIVITAS[1], and developed within the INCOME[2] project (Multiscale Context Management for the Internet of Things), where we focus on context management for mass market context-aware applications [5], [6]. The context management middleware is a highly heterogeneous and dynamic component-based multiscale system. Its components can be as various as ontological knowledge bases, social network analysis components, GUI components, low-level sensors. They must be dynamically deployed at different levels of the system, from communicating objects to cloud computers. In this project, we aim at developing a framework for the deployment of multiscale context managers and beyond of multiscale systems. The following scenario depicts some specific parts of a multiscale system, which will be used to illustrate our proposition throughout this paper.

City of Toulouse, France, 2015. The IT Department of the city recently invested in a system called "MultiModal Mobility in MultiscalE" (4ME). Its purpose is to provide citizens with a multimodal public transport control and guidance system, which aggregates data sources and services from institutional and professional providers all around the city. Offered services are for example bus itineraries, timetables and their real-time positions, bike availability in bike sharing stations, traffic alerts, parking system information. Users of this system are equipped with a smartphone connected to a network. They can benefit from dynamic guidance services, which customize their trips according to a large set of pieces of information and events (e.g., weather conditions, night vs day, preferred itineraries collected by social networks, wish to meet friends on the road).

This multiscale system mainly contains three families of software components. At the lower level, there are components dedicated to context acquisition, such as context data collectors that should be deployed in each bus, in each bike sharing station, or in each parking. Next, there are infrastructure components, such as middleware components that transport data and events between components, or components that filter and aggregate data to infer more abstract information. At the last level, the system includes business components that comprise for example the graphical user interface for the clients, or components of the route planner that should run on some cloud computers.

These various components should be deployed on many devices (from smartphones to cloud computers). They communicate with each other on different networks (from personal area network to the Internet). They must be deployed either over the whole city or at a short distance from a given place. They allow different actors to interact (from single users to communities of users). Their host devices belong to different administrative entities. Thus, this system is multiscale in the device, network, user, geography and administration viewpoints.

Within the motivating scenario, we focus on the deployment of some 4ME software components, which highlight the interest of multiscale deployment properties. The deployment process of such a system should consider a dynamic deployment domain. Moreover, multiscale properties related to the deployed components and to the deployment domain should be expressed by the deployment designer. For instance, a context management component that contains probes counting available bikes must be deployed on each bike sharing station. For load sharing purpose, a component that provides the bike availability service must be deployed every five bike sharing stations on a device connected to a WiFi network in order to distribute the computational load close to the users. A component offering the route planner must be deployed on a cloud computer hired by the city. A middleware communication component that transfers historical data to the cloud

[1] http://www.civitas.eu/content/public-transport-control-and-guidance-system
[2] http://anr-income.fr

must be deployed on each local network. For each smartphone in the group of 4ME users arriving in the city, a graphical user interface component dedicated to the services currently available in the city must be deployed. The system should be able to catch any new device appearing during the lifetime of 4ME. To enrich the service, a social network component should be deployed on one smartphone chosen among the smartphones of each group of friends. This smartphone should have at least a CPU of 1Ghz, and 1Mb of available memory. This component has to be always reachable. If ever the component or the smartphone becomes unreachable, the component is dynamically redeployed on another smartphone of the group. Finally, a business component that stores the history of messages exchanged on the social network must be deployed on the cloud owned by the bike sharing service of the city.

III. Multiscale deployment approach

In this section, we justify the need for software autonomic deployment in multiscale systems (Section III-A), then we exhibit multiscale deployment features from the scenario of the previous section (Section III-B), and discuss the need for a domain specific language. Finally, we introduce our approach for multiscale software deployment (Section III-C).

A. Autonomic software deployment

The traditional way of deploying a software system consists in defining a static deployment plan, enumerating which components have to be deployed on which devices. It is often made by a human operator. Multiscale systems, come with high dynamics, devices availability concerns, topology evolutions when new devices appear, and others disappear due to disconnections or failures. With multiscale systems, the deployment domain has to be discovered just in time (*i.e.*, just before application runtime). Thus, the deployment plan must be continuously adapted to take into account the instability of the network of machines (*i.e.*, connections and disconnections), mobility and openness, and to variations of the availability and of the quality of the resources. Moreover, a high number of devices and of software components (with their different versions) is involved in multiscale systems. Automation and autonomy in deployment management is required again.

Traditional deployment is not an option anymore. An autonomic deployment middleware is required to be able to adapt at runtime, in an uninterrupted process, the deployment plan to the effective deployment domain topology and its properties. Indeed, the autonomic computing approach [7], where the system self-manages some properties (both for self-configuration and self-healing purpose) provides interesting options for multiscale distributed software deployment. This is what we call "*autonomic deployment*".

Appropriate methods and tools are necessary to design, control and automate the deployment process. For the deployment design, we propose to describe the *deployment properties*, coming from different stakeholders, considering two different sets: *deployment requirements* concerning the architecture of the system and *constraints*. As an example of deployment requirement, the deployment designer may want that a given component should be installed on every smartphone of a given geographical area, or that two components should be deployed on two different devices connected to a same local network. In addition, as an example of a constraint, the designer may specify that the mobile devices considered should run Android, have an active GPS, or be connected by WiFi. All these properties should be gathered at design time by the deployment designer.

The deployment middleware ensures the satisfaction of the requirements and constraints: at deployment design time, it computes and realizes a deployment plan that satisfies all of them (if possible) then, it checks the properties during the whole life-time of the system, handles the deployment and adapts the plan after failures or changes in the topology. Note that presenting the way to handle and adapt the plan at runtime is out of the scope of this paper.

B. DSL for multiscale deployment design

To support the description of the deployment properties, the existing platforms propose formalisms such as architecture description languages, XML or DSLs. DSLs present several advantages: they use idioms and abstractions of the targeted domain, so they can be used by domain experts; they are light, easy to maintain, portable, and reusable; they are most often well documented, coherent and reliable, and optimized for the targeted domain [8], [9], [10].

We advocate for a DSL dedicated to the expression of multiscale deployment properties. In this paper, we propose MuScADeL. It must answers to multiscale requirements, and thus allows the deployment designer to express:

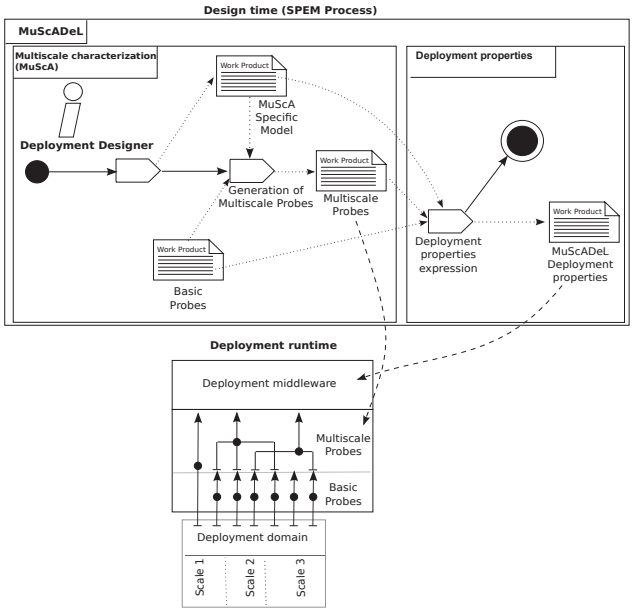

Fig. 1. Overview of the MuScADeL process

R1 Device requirements in terms of scales (e.g., of local memory, storage capacity, processing power or network connectivity);
R2 Network or geography requirements in terms of the smallest scale shared by two devices (e.g., the two considered devices are required to be in the same local area network, or in the same city);
R3 Deployment requirements concerning the network, geography, user and administration scales to which the deployment devices should belong (e.g., deploy a component on the smartphones that arrive in the city);
R4 Deployment requirements to specify ratios of components (e.g., deploy one component A for every four components B).

With this DSL, the deployment designer can describe deployment properties from which a deployment plan can be computed. The computation of the deployment plan is done during the deployment runtime by a constraint solver that also needs as input the list of all available hosts and their properties.

In order to do that, the deployment system has to handle a system of probes that collects all the required information about the hosts. Probed data range from concrete system properties such as free RAM to more abstract multiscale properties such as dimensions and scales. As an example, a multiscale geographical probe with the three scales of city, country and continent can give the information for the underlying device for each scale if available.

C. Overview of the multiscale deployment design process

Fig. 1 shows the general approach we adopt for the deployment design process. The process is detailed with a SPEM [11] diagram. It is divided into two main steps. In the first step, we use the MuScA metamodel and a multiscale characterization process in order to define the multiscale nature of the deployment domain (select viewpoints, dimensions and scales) with a multiscale model. This model is then used through a model driven approach to generate multiscale probes, which consolidate data given by lower level probes, called basic probes. This first step of our approach is detailed in Section V. The second step aims at expressing the deployment properties using our DSL MuScADeL, described in Section VI. By using the MuScA specific model and the multiscale probes obtained in the first step as a support of properties expression, we are able to define a deployment policy that is naturally multiscale aware. During deployment runtime at the right of Fig. 1, the deployment middleware use the data collected from the probes installed in the deployment domain to deploy and adapt the software distribution.

IV. Related works

In this section we present an overview of related works on software deployment that propose the use of a DSL. Additionally we examine how some multiscale concerns take place to a certain extent in existing solutions for software deployment.

To facilitate the deployment designer's work, Dearle *et al.* [12] define the DSL Deladas. The designer specifies the deployment domain and its characteristics and a set of deployment properties (called constraints). From this specification, the framework generates a deployment plan. The constraint-based approach frees the deployment designer from specifying exactly the location of each component, and from rewriting the plan in case of problems with a resource. Deployment is resilient: at runtime, when the deployment middleware detects a constraint violation, it tries a local repair if possible. Otherwise, a new plan is generated and then executed by a "satisfy/enact" component without human intervention. However, the management is centralized and openness is not taken into account, because the set of hosts being statically defined in a file by the deployment designer. Besides, Deladas does not allow the designer expressing multiscale properties.

Matougui *et al.* [13] present a middleware framework designed to reduce the human cost of setting up software deployment and to deal with failure-prone and change-prone environments. This is achieved by the use of a high-level constraint-based language and an autonomic agent-based system for establishing and maintaining software deployment. In the DSL j-ASD, some expressions dedicated to deal with autonomic issues are proposed. But, they only target large-scale or dynamic environments (such as grids or P2P systems), at the same network scale.

Sledviewsky *et al.* [14] present an approach that incorporates a DSL for software development and deployment on the Cloud. Firstly, the developer uses a DSL in order to describe a model of the application. Secondly, this DSL code is translated into a specific code in charge of the automatic deployment onto the Cloud. Authors highlight the need to facilitate the work of the deployment designer, and that using DSL is a solution for that. Nevertheless, this approach is specific to deploying on the Cloud.

When we have considered existing works on software deployment, we also tried to figure out if some multiscale concerns could be expressed. Considering the network point of view, we noticed that there is actually no tool that handles more than one kind of network (most of them target local networks). From the device point of view, deployment solutions such as the one presented in [15] for Cloudlet, or Kalimucho [16], which both allow to transfer computations from devices with limited capacity to powerful computers, handle devices at different scales. But most software deployment tools are specialized for one kind of devices: personal computers in most cases, smartphones, etc. From the geography point of view, some technologies such as Codewan [17], Kalimucho, or Cloudlet basically take into account the proximity between hosts. So, these technologies can be considered for deploying software for multiscale systems in the geographical viewpoint. But in practice, proximity is more related to network properties than to geographic location.

The result of this study shows that there is a lack of expressiveness in existing deployment languages for defining various multiscale properties.

V. Scale-awareness framework

In this section we present the MuScA framework. We present the model driven approach, detail the MuScA metamodel, and its use through an example of MuScA model and generated artefacts.

A. Model driven approach

The concept of multiscale viewpoints has been introduced in Section II-B as a way to analyze the multiscale nature of the motivating scenario. Our experience shows that each system may be analyzed through different viewpoints and scales. Each characterization defines a multiscale vocabulary for the characterized system. It can be used by the deployment designer in MuScADeL and leads to the generation of specialized probes used for scale-awareness purpose at runtime.

In order to formalize the multiscale characterization process and to use it in the deployment design, we have chosen to follow a model driven architecture (MDA) approach (using the four OMG meta-modeling layers [18]). We define the MuScA metamodel (M2 level) with the Ecore meta-metamodel [19] (M3 level). The classes of the MuScA metamodel represent multiscale concepts. With MuScA, we are able to define characterization models (M1 level). This characterization may be used for one or several real world systems (M0 level). We also follow the model driven approach in order to automatically produce artefacts, for instance probe artefacts for scale-awareness purpose.

B. MuScA metamodel

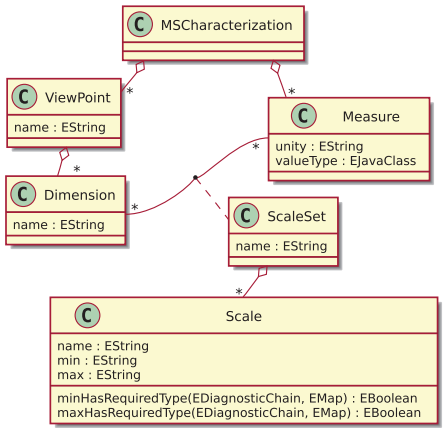

Fig. 2. MuScA: Multiscale characterization metamodel

The MuScA metamodel is shown in Fig. 2. This metamodel is based on the vocabulary used in the multiscale characterization process —*i.e.*, multiscale characterization, viewpoint, dimension, measure, scale set, and scale. An instance of *MSCharacterization* is the result of a characterization process. A characterization considers several *ViewPoint*s (e.g., at M1: *Geography*, *User*, *Device*, and *Network* viewpoints). Each viewpoint determines a restricted view of the studied system. In a given viewpoint, the view of the system is studied through several *Dimension*s, which are measureable characteristics of the elements of the view. For example, for the *Device* viewpoint, the system devices can be analyzed through the *StorageCapacity* (M1) dimension. As previously mentioned, a *Dimension* is measurable, which means it can be associated with one or several *Measure*s. For example, at the M1 level, the *StorageCapacity* dimension may be measured with the *Bytes* measure or the *KiloBytes* measure. For the association of one dimension with one measure, the designer defines a *ScaleSet*, which is an ordered set of scales relevant for the studied system. For numeric measures, a *Scale* is defined by its min and max bounds. For some viewpoints, the system may present several instances of one scale. For example, if we take the *Geography* viewpoint, in the *Administrative* dimension, the *Town* scale (M1) has several instances (M0 level) —*i.e.*, the different towns where entities of the system are present.

C. MuScA model as a multiscale characterization

In this section we apply a multiscale characterization process to the scenario described in Section II-B in order to build a MuScA model of the scenario. The deployment domain of the scenario is multiscale in the device, administration, network, and user viewpoints, but we only describe the device and the administration viewpoints. Firstly, we decide to study the device viewpoint through two dimensions: the storage capacity measured in bytes (numeric measure) and the type of device measured in what we call the "device type measure" (semantic measure). For the type dimension, we identify three scales (hidden in the figure): smartphone, cloudlet, and cloud. For the storage capacity dimension, we also identify three scales: kilobytes, gigabytes, and petabytes. Secondly, we study the administration viewpoint through the administrative level dimension measured in what we call the "administration level measure" (semantic measure). We identify three scales: team, service, and enterprise.

D. MuScA probes

With a MuScA model, we automatically produce artefacts for scale-awareness purpose. In particular, we generate software artefacts called multiscale probes used in the deployment middleware. These probes are monitoring programs that are to be deployed on each device of the deployment domain in order to return its associated scale for a specific scale set —*i.e.*, for a specific dimension associated to a measure in a viewpoint. We generate one probe per viewpoint, and each probe exposes one method by dimension that returns a scale. As mentioned in Section V-B, for some viewpoints, there may be several scale instances for a scale. For example, the *Town* scale has as many instances as real world towns involved in the system. Therefore, the multiscale probes of these viewpoints contain one more method per scale set that returns the scale instance associated to the probes's device for a specific scale. These probes and the generated methods can be completed to implement a specific logic, in particular to call basic probes, as shown in Fig. 1, or to implement specific semantic measures.

E. MuScA achievements

We have implemented MuScA with the Eclipse Modeling Framework Project[3] (EMF). The MuScA metamodel is defined as an instance of the Ecore metametamodel. EMF generates a specialized model editor, which can be used to build the MuScA model presented in Section V-C. Then, we use the Acceleo[4] code generator to produce multiscale probes, which are implemented in Java. We have currently implemented device and geography multiscale probes.

VI. MuScADeL

In this section we detail our proposition of a DSL dedicated to the autonomic deployment of multiscale distributed systems, named MuScADeL, and the use of MuScA in it.

A. Elements of the language

With MuScADeL deployment designers may express multiscale deployment properties in terms of both designer requirements and components constraints. It is possible to express the deployment properties of a monolithic or of a huge component-based software on a deployment domain, which can be composed by one to thousands of devices. MuScADeL provides abstractions and can be used without a strong expertise in the realization of the deployment activities. The MuScADeL grammar has been defined in EBNF syntax[5].

For the presentation of the language, we rely on the scenario presented in Section II-B. The code is presented in five extracts, which respectively present components (Listing 1), criteria (Listing 2), probes (Listing 3), multiscale probes (Listing 4), and deployment requirements (Listing 5).

The unit of deployment is the component. The MuScADeL code lists the component types of the system (keyword **Component** in Listing 1). A Component description has several fields. The **URL** field specifies the address where the component is reachable for download. The **DeploymentInterface** field specifies the interface of the component, that is necessary for the interactions with the deployment system since the latter must configure, start, manage, and stop the component. The **Dependency** field lists required components: when installing the component, the deployment system checks that whether the required components are installed, or if not, installs them. Constraints can be given when specifying a component. The **Constraint** field lists hardware and software criteria defined using the keyword **BCriterion**, see line 4 of Listing 2 — **Bcriterion** stands for basic criterion in opposite to multiscale criterion, cf. VI-B — and that the component must satisfy. By default, these constraints are permanent, *i.e.*they must be satisfied both when generating the deployment plan and at runtime, so that the deployment system must check that there is no constraint violation at runtime. Constraints can also be defined in order to be satisfied only when generating the initial deployment plan and disregarded at runtime: the keyword **InitOnly** allows to specify this kind of constraint.

```
1  Component SocialNet {
2    Version 1
3    URL "http://income.fr/4ME/SocialNetwork.jar"
4    DeploymentInterface fr.income.4me.DISocNet
5    Constraint CPURAM
6  }
7  Component History {
8    Version 1
9    Dependency FileManagement
10   URL "http://income.fr/4ME/History.jar"
11   DeploymentInterface fr.income.4me.DIHistory
12   InitOnly Constraint MaxFile
13 }
```

Listing 1. **Component** type definition in MuScADeL

Criteria are named and specified using the **BCriterion** keyword. A criterion is a conjunction of conditions concerning probed values, like in CPURAM (Listing 2, line 4). There are two kinds of conditions concerning either the existence or liveliness of a probe, or a specific value given by a probe. In the first case, the condition is composed by the probe name and the keywords **Exists** or **Active**, which are defined for any probe interface. For example, in Listing 2, at line 3, the used probe is Wifi, and the condition uses default methods **Exists** and **Active**. If it is a value condition, it is composed by the probe name, a method call, a comparator, and a value. In this case, the method is probe-specific, and defined in the probe interface. For example, in Listing 2 at line 6, the used probe is RAM, the information method used is Free, and its value is compared to the number 1, for 1Mb. A criterion can be used to define both a component constraint (cf. Listing 2, line 5) or a deployment requirement (cf. Listing 5, line 5).

```
1 //Maximum file size must be at least 2G
2 BCriterion MaxFile { FileSize.Max > 2; }
3 BCriterion WifiActive { Wifi Exists, Active; }
4 BCriterion CPURAM {
5   CPU.Proc > 1;  // at least 1Ghz of CPU
6   RAM.Free > 1;  // at least 1Mb free RAM
7 }
```

Listing 2. **BCriterion** definition in MuScADeL

Probes are defined using the keyword **Probe**, with two mandatory fields: **ProbeInterface** and **URL** (cf. Listing 3).

```
1 Probe FileSize {
2   ProbeInterface fr.income.MaxFileSize.DIimpl
3   URL "http://income.fr/4ME/filesize.jar"
4 }
```

Listing 3. **Probe** definition in MuScADeL

The code in Listing 4 was generated by MuScA and integrated when the deployment designer chose a multiscale characterization model. It defines multiscale probes using the keyword **MultiScaleProbe**. A specific keyword is necessary because basic and multiscale probes are considered in a different way when generating the deployment plan (see Section VI-B). As **Probe**, **MultiScaleProbe** has two fields: **MultiScaleProbeInterface** and **URL**. At runtime, a multiscale probe allows to identify the scale of their host device in a given viewpoint/dimension/measure .

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

[4]http://www.eclipse.org/acceleo/

[5]The full grammar is available at http://www.anr-income.fr/T5/muscadel-ebnf.html

```
1  MultiScaleProbe Admin {
2    MultiScaleProbeInterface
3      fr.income.AdministrationProbeImpl
4    URL "http://income.fr/AdminProbe.jar"
5  }
```

Listing 4. `MultiScaleProbe` definition in MuScADeL

Once all of these elements have been specified, the deployment properties of the overall multiscale system can be expressed. The operator `@` allows to specify requirements specific to a component. The overall properties can take several forms as it is illustrated in Listing 5. We describe below each deployment requirement and link it to the requirements expressed in Section III-B:

```
1  Deployment {
2    //the bike sharing station
3    ContextMan @ Device.Type.Cloudlet,
4       Admin.Level.Service("Toulouse.SharingBikes");
5    BikeAvail @ 1/5 ContextMan, WifiActive;
6    RoutePlanner @ Device.StorageCapacity.Giga,
7       Admin.Level.Entreprise("Toulouse");
8    Comm @ Each MSNetwork.NetworkRange.LAN;
9    GUI @ All, Device.Type.Smartphone;
10   SocialNet @ All, User.NumberOfUsers.Group;
11   SocNetHist @
12      SameValue User.NumberOfUsers.Group(SocialNet);
13   History @ Device.StorageCapacity.Giga,
14      Admin.Level.Service("Toulouse.SharingBikes");
15   Stat @ 5..10, Device.Type.Cloudlet;
16 }
```

Listing 5. `Deployment` requirement definition in MuScADeL

The component `ContextMan` must be installed on all the devices that *(i)* have the scale `Cloudlet` on the dimension `Type` of the viewpoint `Device` (requirement R1 — line 3) *(ii)* and are administrated by the service `"Toulouse.SharingBikes"` (requirement R3 — line 4). Components `BikeAvail` must be deployed on devices satisfying the basic criterion `WifiActive`, the ratio expression (requirement R4) `1/5` specifying that there should be one `BikeAvail` component deployed for five `ContextMan` components (line 5). One component `RoutePlanner` must be deployed *(i)* on the scale `Device.StorageCapacity.Giga` (requirement R1) and *(ii)* in the city of `Toulouse` (requirement R3 —line 6 and 7). Components `Comm` must be deployed on one device of each local network area (LAN) (requirement R3 — line 8). The component `GUI` must be deployed on all devices of the scale `Device.Type.Smartphone` (requirement R1), *i.e.*, on all the smartphones of the domain (line 9). The component `SocNetHist` (which keeps a history of the social network chatroom) must be deployed on a device that belongs to the same group as the device on which the component `SocialNet` is deployed (requirement R2 — line 12). The component `Stat` (which calculates statistics on bike use) must be deployed on `5` to `10` devices (requirement R4) of the scale `Device.Type.Cloudlet` (requirement R1 — line 15).

The keyword **`DifferentValue`** allows to specify the contrary of **`SameValue`**. Using these keywords, it is possible to define a requirement concerning a scale. For example, `Comp` **`@ SameValue`** `Device.Type(Comp2)` expresses that the component `Comp` must be deployed on a device that has the same type as the component `Comp2`.

Some constructions of the DSL are particularly dedicated to the expression of properties related to dynamics and openness. We have already mentionned that, by default, constraints on the deployed components have to be satisfied at runtime. Besides, when specifying the **`Deployment`**, the keyword **`All`** states that a component should be deployed on a domain, even if the domain evolves dynamically, that is to say on devices entering the domain (and considering those leaving it). In the example, the component `GUI` should be deployed on every smartphone of the domain, including those that enter in the domain after activating 4ME application; so, the deployment plan evolves dynamically according to entering and leaving devices.

B. MuScA in MuScADeL

As shown in the previous code, deployment designer requirements may include multiscale related requirements. As a MuScADeL code is linked to a MuScA specific model, the MuScADeL editor can check that dimensions and scales conform to the ones defined in the MuScA model associated with it. In addition, multiscale requirements are verified at runtime by the multiscale probes generated for this MuScA model.

Fig. 3 is a UML class diagram that summarizes the MuScA and MuScADeL metamodels. Only some parts of the models are shown in order to highlight the links between them. The MuScADeL metamodel is limited to the criterion part of the deployment requirement.

In the MuScADeL metamodel, *Criterion* is specialized in *MSCriterion* to express multiscale criteria. This element is specialized in *CompValue*, *Each*, and *Simple* to handle the different multiscale deployment requirements policies expressed in Section VI-A (multiscale criteria appear in such a way in the MuScADeL code, cf. Listing 5; contrary to **`BCriterion`** there is no keyword to define them). For example, *Each* stands for "on each device". The expression of a multiscale criterion can concern either a *Scale* (M1 level) or a *Scale Instance* (M0 level). For example, we can express the deployment of a component on each LAN (M1 level) or on a specific LAN (M0 level).

This link between MuScA and MuScADeL insures a correct use of the multiscale probes in the expression of deployment requirements.

C. MuScADeL achievements

Using Xtext and Xtend frameworks[6], we have realized an Eclipse plugin for the edition of MuScADeL. Using Java and Eclipse makes MuScADeL editor multi-platform compliant and easy-to-use for the deployment designer. Moreover, it runs alongside MuScA, allowing the deployment designer to be able within the same engineering tool to define new multiscale viewpoints, dimensions or scales, before using them in the deployment DSL.

[6]www.eclipse.org/Xtext

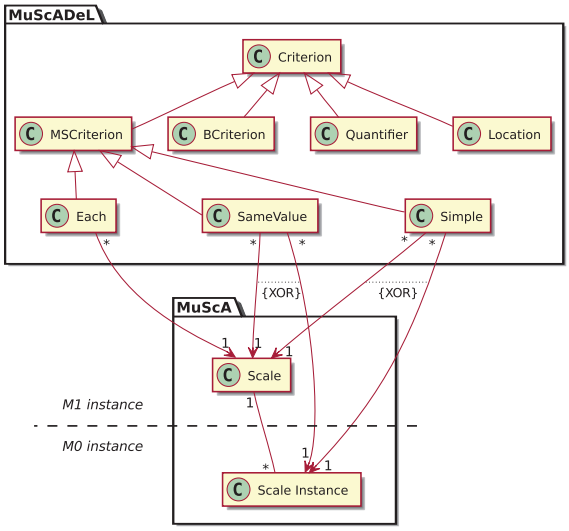

Fig. 3. Part of the MuScADeL metamodel, including MuScA use

VII. Conclusion

Autonomic software deployment is essential to install up-to-date multiscale distributed systems. Over the past decade, some deployment frameworks for distributed systems have emerged. They have limitations in terms of the targeted deployment domain, and in term of autonomicity. In this paper, we have presented MuScADeL a domain-specific language for autonomic deployment of multiscale distributed systems such as those experienced with the IoT.

MuScADeL allows designers to abstractly define multi-scale deployment properties without exact knowledge of the deployment domain. MuScADeL properties are expressed in accordance with the multiscale characterization of the system to deploy. The multiscale nature of a system is defined with MuScA, a framework to identify the relevant viewpoints, dimensions, measures, and scales for a given multiscale system. MuScA has been used with the INCOME project for software infrastructure for context management in the context of the IoT. This project targets the deployment of mass market smart applications. For this project, we have analyzed many scenarios and use cases in ambient systems. With the characterization process, we have been able to select, among existing ones, the viewpoints, dimensions and scales relevant for a given system, or to define new ones. Each characterization enables the framework to extend its multiscale vocabulary and its multiscale probes. Thus the framework learns and memorizes new viewpoints, dimensions and scales to be proposed for the next characterization.

Our contribution has been validated through a complete chain of frameworks. Specialized editors allow designers to define and validate multiscale characterization and multiscale deployment descriptors. MuScA generates multiscale probes for scale-awareness purpose. This chain will be used by multiscale deployment middleware that apply deployment properties defined with MuScADeL.

Acknowledgment

This work is part of the French National Research Agency (ANR) project INCOME [5] (ANR-11-INFR-009, 2012-2015). The authors thank all the members of the project that contributed directly or indirectly to this paper.

References

[1] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The Future Internet*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6656, pp. 431–446.

[2] M. Presser and S. Krco, "D2.1: Initial report on IoT applicationsof strategic interest," The Internet of Things Initiative, Jul. 2011.

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

[4] ISO/IEC/IEEE, "Systems and software engineering — Architecture description," ISO/IEC/IEEE Joint Technical Committee, International Standard ISO/IEC/IEEE-42010:2011, Dec. 2011.

[5] "INCOME," http://anr-income.fr, February 2012, Last access 2013.

[6] J.-P. Arcangeli, A. Bouzeghoub, V. Camps, C. M.-F. Canut, S. Chabridon, D. Conan, T. Desprats, R. Laborde, E. Lavinal, S. Leriche, H. Maurel, A. Pninou, C. Taconet, and P. Zarat, "INCOME - Multi-scale Context Management for the Internet of Things," in *Ambient Intelligence, 3rd Int. Joint Conf. AmI 2012*, ser. Lecture Notes in Computer Science, vol. 7683. Springer, 2012, pp. 338–347.

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

[8] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," *ACM Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[9] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Software: Practice and Experience*, vol. 39, no. 15, pp. 1253–1292, 2009.

[10] J.-P. Tolvanen and S. Kelly, "Integrating models with domain-specific modeling languages," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ser. DSM 10. ACM, 2010, pp. 10–1. [Online]. Available: 10.1145/2060329.2060354

[11] Object-Management-Group, "Software & Systems Process Engineering Metamodel (SPEM) v2.0," formal/2008-04-01, april 2008.

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

[13] M. E. A. Matougui and S. Leriche, "A middleware architecture for autonomic software deployment," in *The 7th Int. Conf. on Systems and Networks Communications (ICSNC12)*. XPS, 2012, pp. 13–20, 12619 12619.

[14] K. Sledziewski, B. Bordbar, and R. Anane, "A DSL-Based Approach to Software Development and Deployment on Cloud," in *24th IEEE Int. Conf. on Advanced Information Networking and Applications (AINA 2010)*. IEEE Computer Society, 2010, pp. 414–421.

[15] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[16] C. Louberry, P. Roose, and M. Dalmau, "Kalimucho: Contextual Deployment for QoS Management," in *Distributed Applications and Interoperable Systems (DAIS 2011)*, 2011, pp. 43–56.

[17] F. Guidec, N. L. Sommer, and Y. Maho, "Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks," *International Journal of Handheld Computing Research*, vol. 1, no. 1, pp. 24–42, 2010.

[18] J. Bézivin and O. Gerbé, "Towards a precise definition of the OMG/MDA framework," in *Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001, (ASE 2001)*, Nov 2001, pp. 273–280.

[19] F. Budinsky, E. Merks, and D. Steinberg, *Eclipse Modeling Framework 2.0*, ser. Eclipse. Addison Wesley Professional, March 2008.