Automating the deployment of componentized systems *

Jesús García-Galán[1], Pablo Trinidad[1], and Rafael Capilla[2]

[1] Universidad de Sevilla
[2] Universidad Rey Juan Carlos

Abstract. Embedded and self-adaptive systems demand continuous adaptation and reconfiguration activities based on changing quality conditions and context information. As a consequence, systems have to be (re)deployed several times and software components need to be mapped onto new or existing hardware pieces. Today, the way to determine an optimal deployment in complex systems, often performed at runtime, constitutes a well-known challenge. In this paper we highlight the major problems of automatic deployment and present a research plan to reach for an UML-based solution for the deployment of componentized systems. As a first step towards a solution, we use the UML superstructure to suggest a way to redeploy UML component diagrams based on the inputs and outputs required to enact an automatic deployment process.

1 Problem context

Software systems need to be deployed and redeployed several times as the environment and context conditions often change. This is particularly important during runtime when a system already deployed changes its current configuration and it has to be redeployed (i.e.: post-deployment reconfiguration). In the era of post-deployment, modern desktop software, mobile applications, autonomic systems, and service-based systems among others, demand continuous changes in deployment activities. Furthermore, the decision to realize an optimal deployment is still challenging. Recent proposals [2, 10] attempted to automate this process based on quality concerns, such as reliability and performance.

Today, complex systems demand automatic deployment capabilities in order to keep the system updated. For instance, pervasive software is often deployed and reconfigured over dozens of embedded devices, or cloud-based systems (e.g.: Software-as-a-Service solutions) demand continuous reconfiguration activities to satisfy new customer's needs and quality concerns (e.g.: workload is moved between servers during system upgrading). Hence, the way to achieve an optimal deployment becomes a major goal. However, there is still a lack of generic approaches that model, from the architecture point of view, the inputs and the outputs used to deploy software automatically.

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and by the Andalusian Government under ISABEL (TIC-2533) and THEOS (TIC-5906) projects.

2 Automatic deployment of component-based systems

Deployment activities need to know which software components will map into the physical model. From our view, we understand this process as a continuous activity where the output of a deployment solution feeds again the process for a new deployment (i.e.: re-deployment).

On the search of a process to automatically deploy a new system configuration, we propose a research agenda that encompasses the following three tasks: (i) Define the inputs and outputs required by a deployment process, (ii) Specify a way to map automatically software components onto hardware nodes, and (iii) Use/build a tool to automate the deployment of componentized systems as a proof of concept to check the feasibility of the solution.

In this paper we focus on the first task. In order to represent the inputs and the outputs for automatic deployment, we will use the following three models to describe the components and behavior of any software system: (i) a model to describe the software, (ii) a model to describe the hardware, and (iii) a model to describe the mappings between the software and hardware models.

From the software architecture point of view, we rely on component-based architectures and systems to modularize and do the required mappings between software and hardware. However, because every context is different, we need to describe which software and hardware constraints and dependencies are needed by a particular deployment solution.

For instance, in the physical architecture we define the non-functional properties of the hardware nodes and runtime environment on which the software runs. These physical properties of the nodes (e.g.: RAM capacity, CPU speed, etc.) are used as constraints for the software components that demand a particular system configuration, as they will drive the final software-hardware configuration. Therefore, our proposed model requires enough expressivity to support all the information required by the mappings between software modules and hardware nodes.

Sample scenario: In figure 1 we describe a deployment scenario to motivate our proposal. It consists of a cloud-based platform where two hardware devices (i.e.: one public from Google and one private using three instances of the AppScale platform) are used to deploy the software. The devices and runtime environment exhibit different non-functional properties. The software to be deployed belongs to transportation management system that uses an API, a client, and a back-end management system. The target system defines the set of constraints over the deployment platform that might change and evolve accordingly to new requirements.

In order to perform an automatic mapping for a given deployment configuration, we model the inputs and the outputs of the process using UML [11] for the following reasons:

1. UML is a standard and widely used notation to describe software systems and supported by many modeling tools because it can describe the architecture of a system from different points of view.

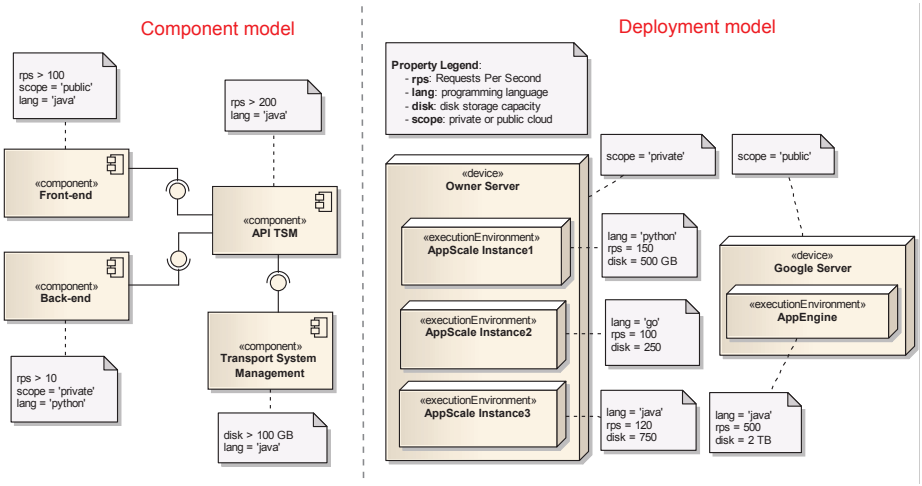

Fig. 1. A Cloud computing deployment scenario for a transportation system

2. It provides a superstructure able to support the mappings between component and deployment diagrams.

3 UML for automatic deployment inputs and outputs

Component and deployment diagrams describing the soft and hard parts of a system can be used to model the inputs and outputs of an automatic deployment process using UML. As a result, an enriched UML deployment diagram shows the allocation of software modules in hardware nodes and according to a set of hardware and software constraints defined in mapping rules.

Components use required and provided interfaces to communicate each other. For instance, service-based systems and cloud platforms often vary the deployment of the running software because changing quality conditions, when the system needs an update, or when existing hardware capabilities have to be extended. Clients that need to rebind to new services dynamically, the reallocation of services in different servers, and new cloud hardware that require to move software from one server to another, are examples of re-deployment scenarios. The component diagram of Figure 1 shows how the *API TSM* component offers two interfaces, the *Front-end* and the *Back-end* that must be allocated in hardware nodes.

In addition, UML deployment diagrams describe a high-level organization of the physical nodes (e.g.: devices, servers, etc.), and according to a particular execution environment and distribution of software modules. The allocation of software elements in the physical architecture rely on mapping rules that are used to automate the allocation process. The right side of Figure 1 displays a deployment diagram with devices belonging to two different execution environments.

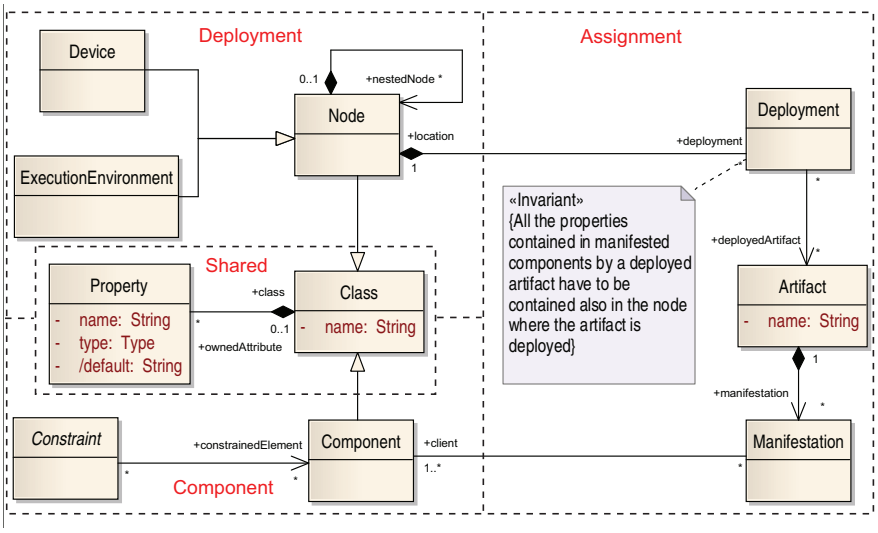

Fig. 2. Deployment with assignment metamodel based on UML superstructure

3.1 The UML Superstructure

The OMG UML Superstructure 2.4 [11] describes the formal specification of UML elements, but in most cases we use only a subset of it when designing with UML elements. In this work we identify the minimum set of elements in the UML supertructure that are needed to map software modules into hardware nodes. These elements can be also extended to provide additional capabilities for automatic deployment. As aconsequence, this superstructure provides a standard way to enhance UML, as we avoid using other approaches (e.g.: such as model-driven engineering) to perform the automatic mappings between UML models.

Figure 2 shows a simplified version of the UML superstructure containing both component and deployment diagrams. The boxes in the *Deployment* area belong to those elements required to define UML nodes in the physical view, whereas the *Component* area contains the elements to design the logical view. Both areas share the properties of the nodes which are used as constraints for the components, such as the *Shared* area displays. Finally, the *Assignment* area encompasses the elements to perform the mappings between the logical and the physical views.

In order to highlight the properties of the nodes, we use the `Property` element to exploit the inheritance relationship between the `Node` and the `Class`. Each property has a name, a type and a default value (e.g.: in Figure 1 we have examples of these properties with their allowed values, such as: rps, lang, disk or scope).

Also, the components model uses the `Constraint` class to represent the restrictions of the nodes (i.e.: properties) and it can be associated to any UML element. These constraints refer to properties that must be defined initially in

the components for deployment purposes. The component model of Figure 1 shows an example of constraints that have impact in the deployment diagram.

In the *Assignment* area, hardware elements are described by means of the `Artifact` class which defines a 1-* correspondence with software components by means of the `Manifestation` class. The deployment of software artifacts in a node is described using a `Deployment` class. In addition, we define an invariant over the Deployment in order to ensure that the components will be deployed only in those nodes that contain the properties that restrict each physical node.

4 Related work

Kruchten [6] pioneered the correspondences between architecture views early in 1995.Other authors like Clements et al. [3] modernize Kruchten's approach to define deployment viewtypes in order to allocate software modules into runtime (additional architecture views can be found in Rozanski and Woods [12]).

Regarding automatic deployment, Kramer and Magee [5] motivate this for autonomic systems while Arshad et al. [1] focus on dynamic reconfiguration using AI planning. Other related works focus on the impact of quality factors for deployment, such as Bushehrian [2] and White et al. [14] which uses performance to compute the nearest optimal deployment using simulation and evolutionary algorithms respectively. In addition, Meedeniya et al. [10] focus on reliability, while Wada et al. [13] focus on SLAs to calculate optimal deployments. Recent approches show examples of automatic deployment using an autonomous engine for service-based systems in the banking domain and using SAT solvers to optimize the best deployment configuration (Cuadrado et al. [4]), while other approaches (Malek et al. [9]) focus on deployment and redeployment activities allocating software components to hardware nodes using a trade-off of QoS properties in order to quantify the quality of the system deployment. All these approaches use proprietary notations or focus on optimization techniques, but none of them exploit the use of the UML superstructure for establishing the correspondence between component and deployment views. Only the MARTE approach (Liehr et al. [7]) exploits the use of UML profiles to allocate models automatically but focused only on UML activity diagrams for real-time execution (RTE) environments.

5 Conclusions and Future work

In this paper we present an attempt of using UML to model the inputs and outputs of an automatic deployment system. This is a first step towards a more ambitious goal of building such system. Next steps lead our future work to focus on: (i) rules to define the mapping process, and (ii) a prototype tool to demonstrate our ideas as a proof of concept using the UML superstructure. Furthermore, we believe that constraint programming can be used to compute the best deployment and model this as an optimization problem.

We are conscious that new requirements might arise in future research that conduct to changes in our proposal of UML-based inputs and outputs. To reduce

the impact of future requirements and in order to increase the confidence on the results presented in this paper, we have built a first version of a prototype research tool. From this validation, some challenging questions have appeared, such as: defining priority levels for hardware constraints, using different optimization criteria, or updating both hardware and software properties after deployment. Our prototype has been built using an MDE approach and currently, it computes basic deployment configurations using constraint programming [8]. We will focus our future efforts in dealing with these questions and tackling next steps.

Bibliography

[1] N. Arshad, D. Heimbigner, and A. L. Wolf. Deployment and dynamic reconfiguration planning for distributed software systems. *Software Quality Journal*, 2007.

[2] O. Bushehrian. Automatic object deployment for software performance enhancement. *IET Software*, 2011.

[3] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2002.

[4] F. Cuadrado, J. Duenas, and R. Garcia-Carmona. An autonomous engine for services configuration and deployment. *Software Engineering, IEEE Transactions on*, 2012.

[5] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. *Computing*, 2007.

[6] P. Kruchten. The 4+ 1 view model of architecture. *Software, IEEE*, 1995.

[7] A. W. Liehr, H. S. Rolfs, K. Buchenrieder, and U. Nageldinger. Generating marte allocation models from activity threads. In *FDL*, pages 215–220, 2008.

[8] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 1977.

[9] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *Software Engineering, IEEE Transactions on*, 2012.

[10] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 2011.

[11] Object Management Group. *UML Superstructure specification*. 2011.

[12] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[13] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. Evolutionary deployment optimization for service-oriented clouds. *Software: Practice and Experience*, 2011.

[14] J. White, B. Dougherty, C. Thompson, and D. C. Schmidt. ScatterD : Spatial Deployment Optimization with Hybrid Heuristic / Evolutionary Algorithms. *ACM Transactions on Autonomous and Adaptive Systems*, 2011.