Analysis of deployment dependencies in software components

Meriem Belguidoum, Fabien Dagnat
ENST Bretagne
France
{firstname.lastname}@enst-bretagne.fr

ABSTRACT

Administration and deployment of software systems become increasingly complex. This complexity results from the need of uniform access to applications from heterogeneous terminals through different communication infrastructures. Moreover, applications consist of complex architectures of lots of small components connected together. A first step to simplify deployment is to have a unified and abstract model for representing deployment dependencies and managing them properly. Therefore, we propose here a deployment model and a conceptual foundation for component installation. Installation dependencies and installation rules are expressed in a logic language.

Keywords

component deployment, installation dependency, deployment model, logic description.

1. INTRODUCTION

The uniform use of applications on heterogeneous terminals and communication infrastructures increases the complexity of software deployment. The deployment process is carried out throughout the life cycle of an application [2]. This cycle covers all management activities from the validation of the application to installation and uninstallation, including configuration.

In this context, we aim to find a solution for autonomic and safe deployment. To reach this goal, we think that it is necessary to:

- take into account the evolution of the system (need for autonomy);
- check and validate the deployment (need for safety).

In addition to these objectives we want to keep:

- generality (not concentrated on a particular model) so that our approach can be usable in all models;
- separation of the dependencies from other deployment information (for example, low level deployment mechanisms).

We present here dependencies related to installation using a general notion of components. Following [6], we define a component as a deployment unit identified by the services that it offers and requires. Our approach expresses dependencies in terms of required and offered services. Thus, it can be applied to every component model that integrates these two concepts.

The paper is organized as follows: first, we present our deployment models in section 2. Section 3 presents our deployment architecture. In section 4, we present the installation processing. Finally, we present related works in section 5 and concludes this article by presenting some future work in section 6.

2. DEPLOYMENT MODELS

Figure 1 depicts our four deployment models: resources, mechanisms, policies and properties. The model of the resources contains the entities which are handled during the deployment and thus answers the questions Which deployed entity? and where deploying entity. The model of the mechanisms answers the question What are the actions to carry out? and contains the operations which act on the resources. The model of the policies contain descriptions of the way in which the choices are carried out and thus answer How to deploy?. Lastly, the model of the properties describes the long-term goals of the system, that represents the response to Why deploying like that?. These deployment models are recursive, i.e. each handled entity is composed of other entities. For example, a computer resource contains software resources which are composed of component resources, etc. Thus, each model will provide a structured and hierarchical description.

3. DEPLOYMENT ARCHITECTURE

To check a deployment, we need to describe all real entities in a deployment: deployment environment, deployed entity and deployment mechanism engine (left side of figure 2) in our formal model. Therefore, the deployable entity and deployment environment are interpreted respectively in our formal world into an abstract dependency description and an abstract context description (right side of figure 2), these descriptions are in a logic language. The interpretations of descriptions and deployment actions from a world to another are represented in the figure by dotted arrows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'06 April 23-27, 2006, Dijon, France
Copyright 2006 ACM 1-59593-108-2/06/0004 ...$5.00.
this moment, the interpretation is manual. In this work, we focus on the reasoning engine which takes both the abstract dependency description and the abstract context description and tests using logic rules if the deployment is possible. If the dependencies descriptions are resolved, the reasoning engine updates the abstract context description. Finally, the decision of the reasoning engine is taken into account by deployment mechanism engine, thus the deployment environment will be updated.

4. INSTALLATION

In our approach, the abstract installation of a component is carried out in two stages (see figure 3). The first one checks if installation is possible (installability) by evaluating component dependencies ($D$) in the environment context ($C$). This stage uses checking rules which ensure the validity of dependencies expression in the context ($C + D$). If the installation is possible, the second stage is carried out using installation rules, it calculates the installation effect on the context ($C + D$). Indeed, the component provided services ($S$) (resp. forbidden services ($F$)) become available (resp. forbidden) in the context.

The checking and abstract installation rules are described in more details in [1].

5. RELATED WORK

Several research are carried out to automate deployment process and dependencies management. We quote only those which are the nearest to our work. In [4], an architecture for the representation of dependencies management in component systems is proposed. The description and the specification of these dependencies are assumed to be already present and consistent while, in our approach, we aim to prove the consistency of the specifications and try to apply them in heterogeneous systems.

The work [5] formalize a configuration problem using a rule language. It concentrates particularly on the semantics of the stable models [3] of logic programs. For the moment, we use a logical formalism without focusing on a particular semantics or a particular model but we are interested on reasoning part.

6. CONCLUSION AND FUTURE WORK

We presented here our deployment models, architecture and a formalization of installation dependencies. Component installation is divided into two stages, checking installability and abstract installation. Each stage is proved by a set of logical derivation rules. A prototype of installation system has been built. We are currently evaluating which extensions are the most important.

This work is in its early phase. The purpose is to make systems increasingly autonomous and safe with the guarantees of quality of service. We are working in three main directions which are: dynamism and evolution of the system, other deployment activities, interrogation of non-functional properties (such as security).

7. REFERENCES