Automatic Synchronization of Allocation Models with Running Software

Marco Konersmann, Jens Holschbach
{marco.konersmann, jens.holschbach}@paluno.uni-due.de
paluno - The Ruhr Institute for Software Technology
University of Duisburg-Essen, Essen, Germany

Abstract

Software models, source code, and deployed software are multiple views upon the same subject. These views are often created and maintained as coupled artifacts, which have to be synchronized. The synchronization can introduce inconsistencies, ultimately leading to errors in the understanding of the software. Deployment models can be derived from the running software, and model changes can be automatically applied to their origin, when the mapping between deployment models and the actual deployment follows known standards. Such an automated synchronization of models and deployed programs can decrease errors in the understanding of the deployment aspect of the software.

This paper presents a prototype that derives Palladio allocation diagrams from running software based on the Java Enterprise Edition via an intermediate language. Changes in the allocation diagrams can automatically be applied to the running software. The extracted diagrams can be the base for performance simulations with the Palladio Simulator.

1 Introduction

Performance simulations are more precise the closer the simulated environment is to the real environment. For simulating the performance of software-intensive systems, a variety of concerns are modelled. One of these concerns is the deployment. Deployment models represent parts of a software, hardware nodes that may run the software, and their interrelationships. They usually describe the actual or the desired deployment of a system.

Architectural concerns such as the deployment, abstract structure, or source code can be seen as multiple views upon the same subject (cf. [2]). Synchronizing these views is possible but problematic, because information is declared redundantly. E.g. an actual deployment and a deployment model both represent common information – how the system is deployed. When multiple representations of the same information are not synchronized, e.g. when a model is outdated, the actual system can be misunderstood. The origin of the deployment information is within the infrastructure: the binaries, the execution platform, and hardware nodes have to exist for running a software-intensive system, in contrast to the deployment model, which exist to give an isolated view of the deployment. A deployment model can be derived from the actual deployment.

In the Palladio Simulator [3], a resource environment diagram describes the hardware nodes available for the system, and their interconnection. An allocation diagram relates instances of component types to these hardware nodes. Deriving a Palladio allocation diagram from the actual deployment allows for preventing misunderstandings of the software, and can therefore be a basis for more reliable performance simulations. This paper presents a prototype[1] for bidirectionally synchronizing deployment models with running software.

In the remainder of this paper, we first describe the concepts and design of the approach. Then the implementation and use of the prototype is described with a small example. Related work is presented before we conclude and describe future work.

2 Concepts and Design

The prototype coordinates the translation between deployment models and actual deployments on execution platforms. As different platforms for actual deployments and different types of deployment models exist, the prototype uses an intermediate language (called *Mapping*) to reduce the number of translations. Figure 1 shows the structure of the classes *Server*, *Application* and *Deployment* of the intermediate language. The class *Deployment* connects each one instance of *Server* and *Application*. The prototype manages lists of instances of these classes.

For the actual deployment of software artifacts, some meta data is necessary, which is often not part of deployment models. The classes of the intermediate language own properties regarding the connection

The work presented in this paper is partially funded by the DFG (German Research Foundation) under the grant number GO 774/7-1 within the Priority Programme SPP1593

[1] `https://s3gitlab.paluno.uni-due.de/ADVERT/deployment-synchronizer`

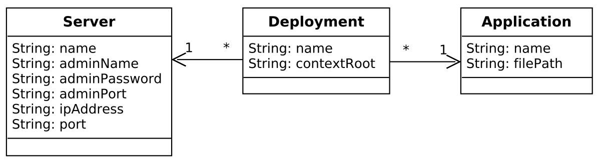

Figure 1: Meta Model for Deployments

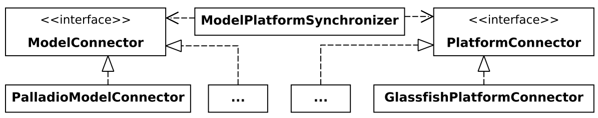

Figure 2: Synchronization Unit and Connectors

to execution platforms, for finding the software artifacts in the file system, and for deploying the artifacts within the right context.

Figure 2 shows the structure of the prototype. The translations are implemented using the interfaces *ModelConnector* for different meta models and *PlatformConnector* for different platforms. The execution of the translations is then orchestrated by the central class *ModelPlatformSynchronizer*, which holds references to implementations of the interfaces. As the translation functionality must be bidirectional, each interface specifies two methods – one for each direction. The *ModelConnector* also has utility methods for handling the model and its editor.

3 Implementation

The approach is designed to be extensible with respect to both meta model and application server. The tool uses the Eclipse platform and therefore uses Eclipse's extension points for extensions with further meta models and platforms. Implementations for adapters for Palladio and GlassFish[2] application servers are provided in the prototype.

3.1 Translation between Palladio and the Intermediate Language

The prototype implements the logic to translate between the intermediate language and Palladio. For translating an intermediate model into an allocation diagram of Palladio (see Figure 3), instances of the class *Deployment* are mapped to *AllocationContexts*. The class *AllocationContext* holds references to *AssemblyContext* and *ResourceContainer*, which correspond to the model elements *Application* and *Server*. The opposite translation works likewise.

3.2 Translation between the Intermediate Language and Deployments

The prototype implements the interface *PlatformConnector* for the GlassFish application server. The

[2]https://glassfish.java.net

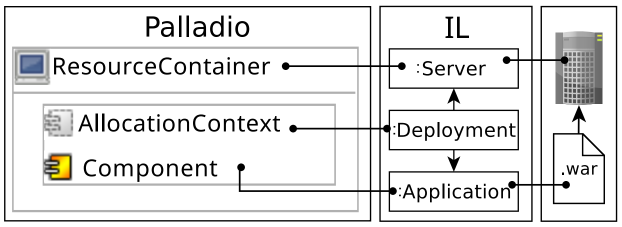

Figure 3: Mapping between Palladio and the Intermediate Language

GlassfishPlatformConnector uses the server's REST API, which allows for a programmatic management of deployments.

In order to update the intermediate model, an HTTP GET request is sent to each server instance to get all contained deployments. The response is parsed and the model is updated accordingly.

To transfer a deployment, that is represented in an intermediate model, to a GlassFish server instance, an HTTP POST request is submitted. The request includes the name of the artifact, the context root, and its enclosed artifact's archive. Then, all deployments in the intermediate model, that have been removed, get removed from the server instance via an HTTP DELETE request including the name of the deployment.

4 Example Use Case

The functionality of the tool is depicted in a small example. A performance engineer found that one of his two servers is under high load permanently while the second one has free capacities. He decides to swap the Java EE components that run on the servers using the presented prototype.

The prototype provides the user interface shown in Figure 4. The figure shows the last of three tabs. The two preceding tabs are used for specifying e.g. implementations of the connectors, the model file, and the meta data for deployments, servers, and applications.

First, a resource environment diagram and a repository diagram has to be available, to know which component types and servers exist, that probably contain

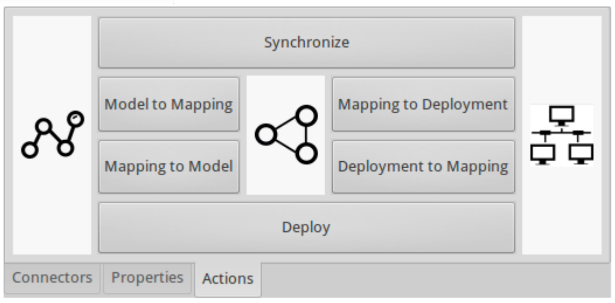

Figure 4: View for Synchronizing Model and Server-Side

deployments. Then, the button "Model to Mapping" must be clicked. This creates the intermediate model based on the resource environment. Meta information about the servers – such as hostnames and ports – can now be entered.

In the next step, the button "Synchronize" has to be clicked to receive information about which deployments are on which server. The *GlassfishPlatformConnector* collects information about deployed .jar and .war files, and relates these deployments to the components in the repository diagram via their names. New *Deployments* in the intermediate model and new *AllocationContexts* in the Palladio allocation diagram have then been instantiated. In this use case one *AllocationContext* will appear on each of the two *ResourceContainers*.

Finally, *AllocationContexts* can be added, modified, or removed. If only existing *AllocationContexts* are changed or remove, clicking the button "Deploy" will update the deployments on the platform. Since in this scenario no further deployments are created, this action will suffice after having swapped the *AllocationContexts*. Otherwise, the intermediate step "Mapping to Model" has to be executed again, because meta data about the new deployments must be added. Then by clicking "Mapping to Deployment", the new applications are deployed on the server.

5 Related Work

The following describes a selection of related work. Some approaches from the reconfiguration domain (e.g. [7]) work with deployment models internally. These models are not made explicit, as they do not expect the deployment to be changed externally.

Brunnert et al. [4] derive performance models from running Java EE components unidirectionally. Deployment models in this approach are always one server with one CPU, where all deployable artifacts are deployed upon.

Approaches for model-based deployment create an actual deployment based on a deployment model. Such approaches can be used to create deployments based on analysis results (e.g. [1]). Other approaches (e.g. [6]) use domain-specific languages for an automated, rule-based deployment. These approaches do not synchronize the model with the actual deployment bidirectionally.

Approaches using run-time models create models of running software. E.g. the iObserve project [5] uses automatically generated, parametrized architecture run-time models. The model used within our approach can be seen as a run-time model. iObserve uses monitoring to build the run-time models. Our approach instead uses a static mapping between the running system and the model.

6 Conclusion and Future Work

In this paper, we presented a prototype that bidirectionally synchronizes Palladio allocation diagrams with the actual deployment on GlassFish servers. Adapters can be built for other modeling languages or platforms. This ensures that the deployment model is not outdated. This enhances the reliability of performance simulations and lowers the risk for misunderstanding the running system. The allocation diagram can be a basis for performance analyses with Palladio, when the other necessary diagrams also exist.

As future work, we plan to integrate the prototype with our tool *Codeling*[3], which can be used to integrate architecture models with source code. When the models are combined, the structure and deployment of software systems can automatically be synchronized bidirectionally. We would also like to extend the approach to handle heterogeneous systems with multiple types of platforms and components. This requires the approach to use parametrizable connector mappings.

References

[1] G. Huang et al. "Towards architecture model based deployment for dynamic grid services". In: *E-Commerce Technology for Dynamic E-Business, 2004. IEEE International Conference on.* Sept. 2004, pp. 14–21.

[2] F. Bachmann et al. *Documenting Software Architectures: Views and Beyond.* Second. Addison-Wesley Professional, 2010.

[3] J. Happe, H. Koziolek, and R. Reussner. "Facilitating Performance Predictions Using Software Components". In: *IEEE Software* 28.3 (May 2011), pp. 27–33.

[4] A. Brunnert, C. Vögele, and H. Krcmar. "Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications". In: *Computer Performance Engineering: 10th European Workshop, EPEW 2013.* Vol. 8168. LNCS. Springer Berlin Heidelberg, Sept. 2013, pp. 74–88.

[5] E. Schmieders, A. Metzger, and K. Pohl. "A Runtime Model Approach for Data Geolocation Checks of Cloud Services". In: *Service-Oriented Computing: 12th International Conference.* Vol. 8831. LNCS. Springer Berlin Heidelberg, 2014.

[6] O. Günalp, C. Escoffier, and P. Lalanda. "Rondo: A Tool Suite for Continuous Deployment in Dynamic Environments". In: *12th IEEE International Conference on Services Computing.* 2015.

[7] R. Abid, G. Salaün, and N. D. Palma. "Formal Design of Dynamic Reconfiguration Protocol for Cloud Applications". In: *Science of Computer Programming* 117 (2016), pp. 1–16.

[3]`https://s3gitlab.paluno.uni-due.de/ADVERT/codeling/`