DECOUPLING CONFIGURATION AND DEPLOYMENT PROCEDURES BY ASPECT-ORIENTED POLICIES

Kurt Englmeier
Schmalkalden University of Applied Science, Faculty of Computer Science, Am Schwimmbad, Schmalkalden, Germany

Ricki Koinig
bwin Entertainment AG, Vienna, Austria

Keywords: Configuration and deployment management, Model-driven-design, Domain-specific languages, Aspect-oriented metamodels.

Abstract: Model-driven development (MDD) has the potential to increase the level of cooperation in software design and adaptation between stakeholders from IT and business domains. Clear and understandable models can raise transparency of business-relevant key characteristics of software.
Our approach addresses a domain-specific configuration language (DSCL) for the dynamic composition and adaptation of applications through configuration information. We concentrate on model representations that reflect individually tailored compositions of generic application modules and their adaptations to individual business needs. Our approach fosters the collaboration in defining application models on two different levels of abstraction. High-level model concepts focus on the definition of process aspects across modules. Low-level concepts cover the complementary role of definition and adaptation of low-level processes that are abstracted away in the high-level concepts.

1 INTRODUCTION

Model-driven development (MDD) (Stahl and Voelter, 2006) fosters increased *responsiveness to changes through more expressiveness and transparency in application design and development.* Both factors help to combine more effectively IT professionalism and domain expertise. The objective is that *human key actors with domain-specific knowledge participate more actively in software development. Domain-specific languages (DSL)* trade generality for expressiveness (Mernik et al., 2005) and make thus feature and process specifics more transparent to non-IT professionals. More transparency, in general, means greater visibility of business relevant software characteristics to application stakeholders.

A generic platform for a specific family of portals (for gaming portals, for instance) consists of a small number of application modules. Changing the composition and configuration of these modules is an easy, albeit limited, but nevertheless powerful way to adapt systems and software behavior to individual needs. More important, it enables organizational decoupling of software development tasks. Organizational decoupling has a clear objective: the platform provider produces a set of generic platforms. The portal providers specify them according to their individual needs, independently from the platform provider. (Anand et al., 2005; Gold et al., 2004) Portal models, as far as they affect individual businesses, are expressed in a representation language that both, the IT experts of the platform provider and the domain experts of the portal provider understand. Our DSCL is inclined to the Configuration Description Language Specifications framework (Bell et al., 2009; Goldsack et al., 2009). The DSCL conceives deployment and configuration (D&C) procedures as modular, self-describing, reusable and tailorable process components (D'Souza and Wills, 1998; Szyperski, 1997) that are combined by descriptions reflecting individual business process aspects (Siobhàn and Baniassad, 2005; Kiczales et al., 1997).

The standardized deployment and configuration

Englmeier K. and Koinig R. (2009).
DECOUPLING CONFIGURATION AND DEPLOYMENT PROCEDURES BY ASPECT-ORIENTED POLICIES.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 321-324
DOI: 10.5220/0002255303210324

language emerges from experiences in configuration and deployment management at bwin Entertainment[1]. Chapter 2 outlines the principles of the DSCL and work related to our approach. Chapter 3 presents the architectural context. In chapter 4, we present the specification of DSCL at different levels of abstraction. Chapter 5 concludes the paper.

2 METAMODELS FOR DEPLOYMENT AND CONFIGURATION ASPECTS

D&C information describes

- deployment processes controlling the composition of generic modules and
- the specification of object states within the modules

in accordance to specific business needs.

Metamodeling is a key aspect of MDD. (Cuadrado and Molina, 2007) A *deployment and configuration metamodel* provides abstract descriptions of deployment procedures, environment configurations and parameter injection. Deployment descriptions ensure the correct deployment of the application modules and their correct configuration in line with the requirements of the target platform. *Parameter injection* addresses the modification of object states at release of the services or during run-time. Injection comprises the two phases, namely *introspection* (retrieval of information about the state of an object) and *intercession* (modification of object state properties). Low-level instructions contain object state templates being completed during deployment or at run-time (see figure 1). *From a different angle, metamodels as well as low-level instructions can be considered as process policies on different levels of abstraction.*

As figure 1 shows, the platform provider pre-defines metamodels and low-level instructions. Many of these definitions thus contain templates that are specified later according to the specifics of the portal provider's IT environment and business needs.

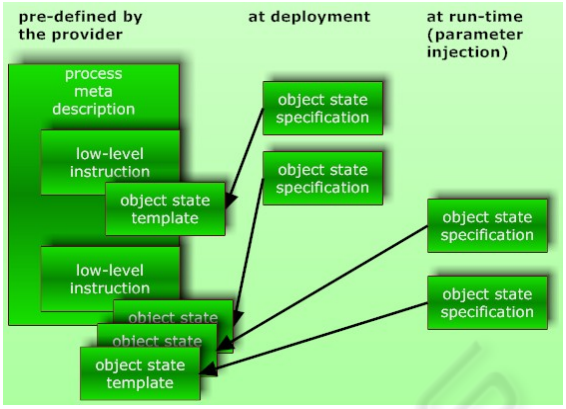

Figure 1: Metamodel descriptions are further specified by low-level instructions. During deployment and later, at run-time, the object states (and partly the low-level instructions) are further specified.

3 ARCHITECTURAL ASPECTS

Composition of the application release package and management of the delivery process are thus platform and content dependent. The deployment target usually comprises a pre-allocated cluster of machines. The cluster can be differentiated into layers along architectural or functional aspects, like a layer for database applications, web front-end applications, back-end applications, etc. A *life-cycle* manager organizes the deployment of the services to one or more specified host platforms (or host layers). It interacts with a deployment portal on the host layer in order to coordinate the deployment. This portal constitutes a service endpoint that is addressed by an Endpoint Reference (EPR). It is thus called *portal EPR*.

The life-cycle manager starts the instantiation of the applications on the deployment target by sending a request to the portal EPR for creating the application instance. The request contains application composition description and configuration information. The portal EPR returns to the life-cycle manager one or more *system EPRs*, that inform about operational characteristics of the host systems or layers.

Afterwards it routes an initialization request to the respective system EPRs. Once all affected applications completed this request, they are *initialized* and the life-cycle manager moves forward to the next life-cycle stage that addresses procedures to put the system into operation (Loughran 2005). The life-cycle manager communicates with an API characterized by the delivery server and the

[1] bwin Entertainment AG produces platforms for game portal providers, primarily gaming platforms.

deployment model it supports. Introspection reflects the actual state of the application objects. For this purpose, it queries the corresponding object property. In turn, the manager can also set properties by sending notifications. It is expected that the deployed application instances communicate with the life-cycle manager throughout their agreed communication channel. They send notification messages to the manager and receive messages through their respective EPR.

4 SPECIFICATION OF THE DSCL

The DSCL consists of two complementary types of semantics: an ontology represent high-level concepts that correspond to the metamodel (see figure 2). It has its focus more on definition and control of processes aspects on a coarse-grained level. *Low-level concepts address instructions that extend and detail high-level concepts* (see figure 3). *Definitions of object state properties base on plain XML* (see figure 4). *The ontology concepts are the building blocks of the D&C metamodels.* Each ontology component is further specified by extensions that refer to low-level instructions. The high-level concept for gracefully shutting down a server instance refers to the generic low-level policy "timePolicy" that is further specified, for instance, in the low-level policy "shutdownTimePolicy".

```
<dco:dcprocedure name="staring & stopping">
  <dco:role id="stop">
        <description>
              All server instances are shutdown gracefully
        </description>
        <policy idref="timePolicy"/>
  </dco:role>

  <dco:role id="start">
  </dco:role>

  <dco:role id="test">
        <description>
              Check server availability
        </description>
        <policy idref="respondPolicy"/>
  </dco:role>
</dco:dcprocedure>
```

metamodel description of a D&C procedure

Figure 2: Part of the metamodel describing a D&C procedure to start and stop server instances. This description is further detailed by low-level instructions.

For semantics representing low-level concepts we use the SmartFrog framework (Goldsack et al., 2009). SmartFrog organizes representations of configuration components in a hierarchical structure with an overlaid naming convention. Figure 3 shows component representations that refer to a template used for the specification of a server instance and for a time policy for shutdown operations.

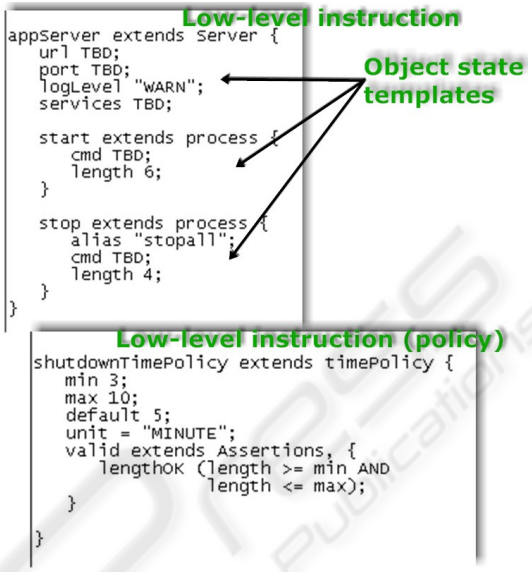

Figure 3: Descriptions of Low-level instructions using SmartFrog.

```
<server id="accounting server">
  <epr id="2"/>
  <url>accounts.node.company.com</url>
  <services>
        <service1 id="install_account"/>
        <service2 id="delete_account"/>
        <service3 id="check_account"/>
        <service4 id="transfer"/>

        <stop cmd="dbctl stop" delay="length"/>
  </services>
</server>
```

specification of object state properties

Figure 4: During deployment or at run-time, object state properties, defined in the templates, are further specified with values specific to the target environment. The snippet here lists the (sub)set of (all available) functions of the accounting application being enabled at this particular server instance.

Besides decoupling and reusability, an outstanding benefit of the language is its simple semantics of extension that is also reflected in our example. Decomposition of deployment procedures is in particular important when specific D&C aspects are better addressed by encapsulated components at different levels of abstraction. Instead of hard-wiring them repeatedly in the different deployment procedures we advocate their separate management by the respective competence team and their loosely coupling in a cross-organizational management of

D&C processes.

The first snippet in figure 3 shows configuration instructions for a portal server instance. The port number will be assigned at run-time, but the level of logging information is already initialized. Further on, two procedures for shutdown and start-up are specified. The server's URL, its port, and the corresponding command strings ("cmd") for the start and stop operations are assigned at run-time. Maximum duration for start-up and shutdown is set six and four time units, respectively. These two parameters are pre-defined in advance. The snippet thus shows an object state template with pre-defined object properties. The second snippet shows a policy that defines time restrictions and preferences for process handling. Naming conventions ensure that the policy is related to the component description when the life-cycle manager initiates the server, starts it, or stops it.

5 CONCLUSIONS

Reusability, tailorability, scalability, and loosely coupling of functional components are objectives barely addressed in current deployment and configuration models and tools. Complexity of applications and the increasing business agility calls for more adaptability and flexibility in all phases of the application development life-cycle (ADLC). We add intensified cross-competence collaboration to these objectives. In many phases, software development can benefit from the active presence of domain knowledge and expertise. We want to endow our customers with the capability to adapt their portals to their individual needs without resorting to IT personnel from the platform provider.

The active role of the customers in D&C opens new business models for platform providers. They concentrate on the development of generic high-performance platforms whilst serving a probably broader market for a particular application family. For the platform providers, the cross-competence collaboration translates into faster adaptation of their products to changing requirements. Eventually, it further translates into shorter time-to-market for new products or for existing products on new markets.

REFERENCES

Anand, S.; Padmanabhuni, S.; Ganesh, J., 2005. Perspectives on service oriented architecture. *Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05)* Vol-2, Orlando (FL), USA, p. xvii.

Bell, D.; Kojo, T.; Goldsack, P., Loughran, S.; Milojicic, D.; Schaefer, S.; Tatemura, J.; Toft, P., 2009. Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation Document. Published at www.ggf.org/documents/ GFD.50.pdf, retrieved March 14, 2009.

Cuadrado, J. S.; Molina, J. G., 2007. Building Domain-Specific Languages for Model-Driven Development. *IEEE Software 24(5)*. pp. 48-55.

D'Souza, D.F. and Wills, A., 1998. *Objects, Components and Frameworks with UML: The Catalysis Approach*, Addison-Wesley, Upper Saddle River.

Gold N.; Mohan, A.; Knight, C.; Munro, M., 2004. Understanding service-oriented software, *IEEE Software 21(2)*. pp. 71-77.

Goldsack, P.; Guijarro, J.; Loughran, S.; Coles, A.; Farrell, A.; Lain, A.; Murray, P.; Toft, P., 2009. The SmartFrog Configuration Management Framework. *ACM SIGOPS Operating Systems Review 43 (1)*, pp. 16-25.

Kiczales, G.; Lamping J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwin, J., 1997. Aspect-Oriented Programing. In: *ECOOP'97-Object-Oriented Programming, 11th European Conference, volume 1241 of Lecture Notes in Computer Science.* Springer, pp. 220-242.

Loughran, S., 2005. Configuration Description, Deployment, and Lifecycle Management. CDDLM Deployment API. Global Grid Forum. Published at http://xml.coverpages.org/CDDML-Deployment-API-SpecificationDraft 20050308.pdf, retrieved April 5, 2009

Mernik, M.; Heering, J.; Sloane, A.M., 2005. When and How Develop Domain-Specific Languages. *ACM Computing Surveys 37(4)*, pp. 316-344.

Siobhàn, C.; Baniassad, E., 2005. *Aspect-Oriented Analysis and Design*. Addison Wesley Professional, Upper Saddle River.

Stahl, T.; Voelter M., 2006. *Model-Driven Software Development*, Wiley & Sons.

Szyperski, C.A., 1997. *Component Software: Beyond OO Programming*, Addison-Wesley, Upper Saddle River.