HAL open science

Model-driven execution of service-based applications

Diana Moreno, Elmehdi Damou

To cite this version:

Diana Moreno, Elmehdi Damou. Model-driven execution of service-based applications. IDM 2011 - Journées sur l'Ingénierie Dirigée par les Modèles, Jun 2011, Lille, France. pp.51-56. hal-00746055

HAL Id: hal-00746055

https://hal.science/hal-00746055v1

Submitted on 9 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Model-driven execution of service-based applications

Diana Moreno-Garcia — Elmehdi Damou

Grenoble Informatics Laboratory - LIG
220, rue de la Chimie
Domaine Universitaire, BP53
38041 Grenoble Cedex 9
{diana-guadalupe.moreno-garcia, elmehdi.damou}@imag.fr

ABSTRACT. *The design, development and execution of dynamic applications is challenging since their execution context is unknown at design-time. In this paper we present a model-driven approach where a service-based application (its goal and properties) is defined at design-time by several models. At runtime, these models are used by our execution platform, the APplication Abstract Machine (APAM), in order to control the application's execution. The APAM delegates the resolution of services to specific managers, in order to ensure/enforce the fulfillment of the application's goal.*

RÉSUMÉ. *Etant donné que le contexte d'exécution d'une application dynamique n'est pas connu à l'avance, la conception, le développement et la gestion de l'exécution de ce type d'applications représentent un véritable défi. Dans cet article, nous présentons une approche dirigée par les modèles dans laquelle une application à services (son but et ses propriétés) est définie à la conception à travers divers modèles. A l'exécution, ces modèles sont utilisés par notre plate-forme d'exécution APAM (APplication Abstract Machine) pour guider l'exécution de l'application. APAM délègue les choix de services à des gestionnaires spécifiques, afin d'assurer/imposer l'accomplissement de l'objectif de l'application.*

KEYWORDS: *service-based applications, MDE, dynamic systems, service execution platforms, evolution, models@runtime.*

MOTS-CLÉS : *applications à services, IDM, systèmes dynamiques, plates-formes d'exécution, évolution, modèles à l'exécution.*

1. Introduction

The Service-Oriented Computing (SOC) approach proposes an interaction protocol in which a client discovers available services from a registry through the service description, and selects and uses the "most appropriate" one (Papazoglou *et al.*, 2007). This approach emphasizes that many implementations and instances of a service may be available, and that the selection of the implementations or instances can be done at any time during the software life-cycle, from design to runtime. Thus, SOC is especially well-suited to build new application types, such as ubiquitous applications where services represent the functionalities provided by devices.

These new types of applications need to use the available services in opportunistic and dynamic ways, and to evolve at runtime. Opportunism means using the services available at runtime, instead of installing and instantiating those planned before execution. Dynamism considers that services can appear and disappear during execution. Evolution considers that new functionalities or requirements can be added at runtime to adapt the application to new execution environments. Controlling the execution and evolution of an application can require re-evaluating decisions taken during design-time. Thus, building service-based applications requires (i) an application description allowing specifying opportunism, dynamism, and adaptation properties, leaving flexibility at runtime, and (ii) a runtime platform that manages the application execution in order to ensure/enforce the fulfillment of its definition (i.e. the goal) despite the environment variations.

Our goal is to extend the usage of software models produced in model-driven engineering (MDE) to runtime environments. Then, we use design models to manage the application execution enabling to perform adaptations, and ultimately to fix design errors or to include new design decisions into the running system.

This paper shows the general principles of our approach. We propose a concept of composite addressing the needs of describing opportunistic, dynamic and adaptive service-based applications. Component selection and adaptation can be performed all along the application life-cycle, ensuring the fulfillment of the application description. We also propose environments and platforms supporting these concepts and mechanisms in the different phases of the application life-cycle, from design to runtime. The paper is structured as follows. Section 2 presents our application model definition. Section 3 shows how an application model is executed by our execution platform for service-based applications. Section 4 presents how the models defining different application properties (concerns) are controlled by specific managers. Finally, section 5 presents the conclusions of our work.

2. Application Model

We consider a software project as a succession of phases whose purposes are to gradually develop and select the application's components. These phases are performed by humans before execution and by machines at execution. In order to

automate component selection at all phases and to control the application execution and evolution, at least its goal and properties must be explicit. Nevertheless, it is difficult to define explicitly at design-time, what a dynamic and opportunist application is. It is due, among other things, to the unpredictable availability of its components (e.g. mobile devices, services launched by third parties). Therefore, the composition (i.e. the exact assembly of components) of such an application, at a given point in execution, cannot be fully anticipated at design-time.

Our approach relies on a composite concept which can describe an application in abstract terms through the goal, properties and constraints that must be satisfied, but also in terms of services and connections. A composite can be seen as the intentional description of the application. We call this description the ***application model***. It can be refined, completed and adapted at any point in the application's life-cycle. This enables a flexible and iterative approach in which some parts of the application are statically selected at design-time, while others are left open for opportunist and dynamic selection at runtime. Execution is seen in our approach as the last iteration, in which selections and adaptations are performed when needed. We argue then that there is a continuum from design-time to runtime, where each point is represented by a composite. All phases share the same service metamodel (Simon *et al.*, 2010), which is abstract enough and independent from specific platforms and technologies. Its main concepts are service Specification, Implementation and Instance. This metamodel was extended (Estublier *et al.*, 2009) to introduce the concept of composite, defined like a constrained service implementation that contains services.

We use SELECTA (Estublier *et al.*, 2009), a constraint language which allows specifying the required characteristics of the composite to build (see figure 1a for a home application example), and an interpreter which analyses the application model and selects components satisfying the constraints to constitute the application. With our approach, properties like opportunism, dynamism, etc., can be specified into specific independent models (see figure 1b for an opportunist model example) for the application component services. Thus, our approach allows mixing different modes of execution (e.g. opportunist and dynamic) for the application components.

```
Composite HomeApp {
 Provides Home;
 Select Thermometer[*](accuracy>90);
 Select Electricity;
 Delay Specification;
}
```

Figure 1a. *Application Model*

```
OpportunistModel HomeApp {
 Thermometer,WaterProbe,ElecProbe;
}
```

Figure 1b. *Opportunistic Model*

At runtime, these models are controlled by the APplication Abstract Machine (APAM) which is our execution platform for service-based applications. Concern-driven managers are in charge of resolving and controlling the application services according to a specific concern (e.g. opportunism, dynamism, distribution, etc.).

3. Application execution

The APAM interprets different application models in order to manage the application execution. It is in charge of selecting the components that are best suited for the application in the current context, and to binding between themselves. The APAM manages two major models (see figure 2): the Service State Model (SSM) and the Application State Model (ASM). The SSM (Simon *et al.*, 2010) describes the "real world"; it represents homogeneously all the services provided by several execution platforms disregarding the technology (OSGi, iPOJO, Web services, etc.). The ASM extends the SSM to manage the concept of composite; it represents the state of an application at a given point in time. Like in (Blair *et al.*, 2009), our APAM model is a model@runtime that provides exact information about the managed application; and the model is causally connected to the real service platforms, allowing application adaptations to be performed at model level.

Executing a composite on the APAM consists in instantiating the different components of the application by resolving the invoked specifications. When resolving a specification, the corresponding implementation and instance objects are created into the ASM. Due to their causal connection, the APAM translates these actions into the "same" actions in the SSM, which has as consequence to deploy, install and start the selected services on the underlying service platforms. The APAM offers the possibility to extend the ASM by adding new concepts and concerns. These new models will be interpreted by specific-concern managers, which will manage the application services involved with a specific concern.

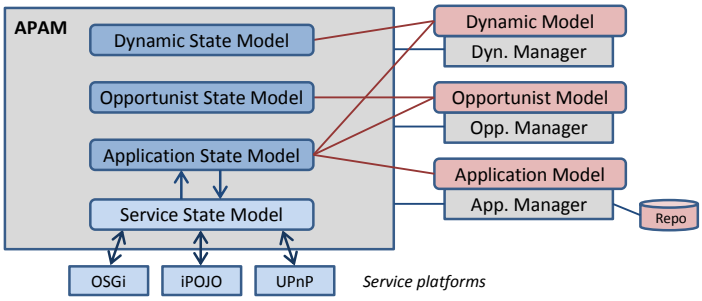

Figure 2. *APAM – General architecture*

4. Application management

The models defined by SELECTA are interpreted by specific managers. The managers are in charge of enforcing the definition of the application for a specific concern. A manager therefore knows and manages its "own" concepts, and maintains a model representing the current state of the involved services. We have identified different concerns – opportunism, dynamism, deployment, distribution,

and so on – and we have defined a manager for each one of them. In the following, we present briefly three of these managers.

– The ***Application manager*** knows and manages the application model. Its goal is to interpret the model and ensure/enforce that the execution conforms to its goal. This manager is unaware of concepts like dynamism, deployment and distribution.

– The ***Opportunist manager*** knows the specified opportunist services, and uses the SSM to select services, when required, from the execution platform.

– The ***Dynamic manager*** knows the concept of dynamic service. It listens to the events from the SSM in order to be aware of the availability of the dynamic components and act on the ASM to bind/unbind services appropriately.

Our managers control and adapt the application to the execution context based on the concern-specific models. This approach based on models at runtime is, for us, an important contribution into the domain attempting to blur the gap between application's design-time and execution (Baresi *et al.*, 2010).

5. Conclusions

Our work focuses on service-based applications, in which the services making up an application can be legacy, heterogeneous, dynamic and potentially shared with other applications. We require application architectures to be extensible in different ways, from supporting different platforms, to adding and managing new concerns.

The work presented here shows our approach to describe and execute an opportunistic and dynamic service-based application. We have shown that such an application definition can be split in independent models, interpreted by independent managers; each manager being much focused and based on a very limited number of concepts, making models compact and simple. We believe that the capability of the system to model the behavior in simple and abstract models is a major contribution.

6. Bibliography

Papazoglou M.P., Traverso P., Dustdar S., and Leymann F., "Service-Oriented Computing: State of the Art and Research Challenges", *IEEE Computer*, vol. 40, 2007, pp. 64-71.

Simon E., Estublier J., and Moreno-Garcia D., "Extensible and General Service-Oriented Platform. Experience with the Service Abstract Machine", *In IEEE Int. Conf. on Service Computing (SCC)*, 2010, Miami, Florida, USA.

Estublier J., Dieng I.A., Simon E., and Vega G., "Flexible Composites and Automatic Component Selection for Service-Based Applications", *4th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2009, Milan, Italy, pp. 6-10.

Blair G., Bencomo N., and France R.B., "Models@run.time", *Computer*, 42(10):22–27, 2009.

Baresi L., and Ghezzi C., "The Disappearing Boundary Between Development-time and Run-time", *In Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER)*, 2010, Santa Fe, New Mexico, USA.