ADAPTIVE SOA INFRASTRUCTURE BASED ON VARIABILITY MANAGEMENT

Peter Graubmann, Mikhail Roshchin

Abstract: In order to exploit the adaptability of a SOA infrastructure, it becomes necessary to provide platform mechanisms that support a mapping of the variability in the applications to the variability provided by the infrastructure. The approach focuses on the configuration of the needed infrastructure mechanisms including support for the derivation of the infrastructure variability model.

Keywords: Service-oriented Architecture, Adaptability, Variability Model, Distributed Systems.

ACM Classification Keywords: C.2.4 Distributed Systems

Conference: The paper is selected from Sixth International Conference on Information Research and Applications – i.Tech 2008, Varna, Bulgaria, June-July 2008

Introduction

Adaptability of IT infrastructures is one of the prerequisites that provides the necessary potential for allowing the realization of application variants (for instance in a software product line context). Usually, application configuration is a task that is driven by application feature selection. The corresponding selection of the appropriate infrastructure is a task derived thereof. It can be supported and automated to a great extent by exploiting dependencies and constraints between variation points and variations in variation models. This is obviously also the case in a SOA context where, as an additional requirement, configuration tasks are expected to be particularly flexible and supposed to be performed during all stages from design until run time.

The central idea of our approach is thus, to use a given configuration, that is, a particular resolution of the variability model of the given product line, and to derive the appropriate configuration of the infrastructure by exploiting dependencies, requirements and constraints described in the variation model.

Automated support of the derivation of the infrastructure configuration assists the application developers in selecting the best fitting infrastructure services and mechanism with appropriate quality of services and relieves them from the burden of investigating infrastructure properties again and again. It facilitates taking into account infrastructure usage patterns and best practices based on the knowledge of infrastructure experts (re-use of infrastructure knowledge). In the case of run time configuration, an automated support becomes inevitable.

Approach

Our approach particularly focuses on the derivation of the needed infrastructure configuration for a given application (that is, a given derivation in a product line). However, in order to provide support for this, an adequate description and a proper formalization of the available diversity of the infrastructure is a necessary prerequisite. Thus, we address the establishing of a variability model (VM) of the infrastructure and concentrate on the derivation of the infrastructure configuration from a given application configuration.

Identifying an appropriate infrastructure configuration is based upon the thorough understanding of the potential for adaptation of the available infrastructure services because they have to be mapped onto the requirements posed by the product variants derived from the given product line. To gain such an infrastructure variability model, we envisage three possibilities:

- The first step is to establish such an infrastructure variability model “manually”; that means that domain engineers knowledgeable of the infrastructure define it in the usual way by identifying the variation points in the infrastructure and the related available features (respective variants).

- The second step is to derive the infrastructure VM from the product line VM. This means to first identify the variation points of the product line that are related to infrastructure issues, then to collect the existing descriptions of the respective infrastructure services and mechanisms (for instance, descriptions of service models, bundles, etc.), and eventually to derive their constraints and dependencies. This results in an infrastructure VM that is precisely tailored to the infrastructure requirements formulated in the product line VM.
- The third step is a combination of the automated and a manual derivation of the infrastructure VM. Here, the challenge is to integrate the delta coming from the domain engineers into the infrastructure VM. A similar mechanism copes with evolutions in the product line and it's VM, adding the thereby emerging delta variability. This approach provides for the evolution of the infrastructure VM.

Having the infrastructure VM in place is prerequisite to identifying the relations between the infrastructure VM and the product line VM which in turn is needed to identify the appropriate infrastructure services/mechanisms from a specific application configuration.

Establishing the relations between the two VMs has to be done explicitly in the initial case where the infrastructure VM is defined manually. In the second case, these relations are established together with the construction of the VM. In the hybrid case, the manually defined VM delta has to be analyzed and the respective relations with the product line VM and its extensions have to be identified.

The derivation of an infrastructure configuration from a given application configuration is essentially based upon the two VMs, the product line and the infrastructure VM, and their relation as established by the previous subtask (in the following, we call this relation VM-relation). It also relies upon the behavior description (probably given as a business process model, etc.) of the given application.

In order to obtain an initial infrastructure configuration, the VM-relation has to be exploited, thereby identifying the specific infrastructure services/mechanisms required by the given application configuration. The infrastructure VM with its own dependency relations and constraints then provides the information for the eventual concrete configuration of the infrastructure.

Reacting to the need of a run time re-configuration of the application follows the same lines; however, the challenge here is to take into account that – since only a part of the original application configuration is changed – only as much modifications of the infrastructure are feasible as absolutely necessary (run time re-configurations have to be as less invasive as possible). For instance, through changing QoS requirements the re-configuration of only the infrastructure becomes necessary. Checking back with the original application configuration (for keeping the constraints and dependencies valid) will ensure the correctness of the revised infrastructure configuration.

Processing Adaptability

Requirements are variable in time by nature. Building SOA-based software product lines relies on the idea to achieve variability and therefore dynamic adaptability. The approach requires that all requirements have to be specified not only – as classically suggested – in a structured list of requirements, but also – as proposed in [Pohl, 2005] – in the form of a variability model. Consequently, the requirements layer has to get extended with an additional formal model, that means that, for instance, groups of requirements R1 and R2 have to be synchronized with Variant 1 and Variant 2, respectively (see Figure 1).

Typical problems, which regularly occur during the life-time of an application, comprise but are not limited to the following cases: data formats or data semantics varying from service to service, unexpected behavior not foreseen during design, missing required functionalities related to implementation errors, altered values of non-functional properties and changing QoS requirements. Adaptability would mean that parts of the software application should be changed in order to cover newly upcoming requirements or to avoid emergent issues. In the business process driven SOA-infrastructure this implies either a change in the part of the workflow, or a substitution of some of the subprocesses with different functionality. Obviously, utilizing the second possibility is

less costly. It can be realized by using already existing processes from other components/services of the related software product line keeping already existing functionality and extending it with required features.

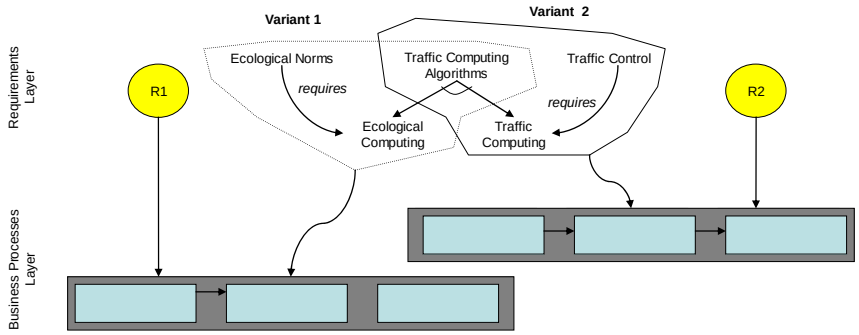

Figure 1. Extension of the Requirements Layer with VM

Managing business process specifications with a variability model can be done using rule-based mechanisms like the Object Constraint Language (OCL), Prolog or others.

A SOA-based infrastructure is built on top of existing services and their models which usually are available in repositories for easy access.

Replacing one subprocess by another is usually not sufficient. The whole dependency chain of related services has to be taken into account (so, a substitute, for instance, may require different protocols and access methods; it may show incompatibility with neighbor services and interface inconsistencies; or different semantics of the input values require further changes or imply the need to find similar services). This means that the whole related information within the infrastructure should be checked and taken into account. Dependency information is specified in form of dependency relations, extensions, and extension points (as defined in OSGi specification [OSGi, 2008]). This allows to extend existing variability models, and thus to provide for more reliable solutions. Such a task can be performed by using existing OSGi and SCA (see [SCA, 2008]) specifications for building dependency graphs, and further transforming that information into a variability model.

Use Case

Consider as an example a large traffic system built on top of a distributed SOA infrastructure. This means there are a lot of participants involved in an application such as permanent entities – traffic lights, cameras and sensors – and temporal entities like cars, bikers and pedestrians, all brought together to interact and to perform various services. The distribution of a SOA infrastructure allows defining particular algorithms for specific places (like townships, crossroads, or sections of a road), depending on the environment and its conditions. A typical task of such a large traffic system is to avoid traffic jams based on the information about the current traffic situation, the traffic density in neighboring roads, the concentration of pedestrians and the local traffic policies (probably including policies to enforce ecological regulations). Each entity can be represented by a service and provides necessary information, like speed of the cars, estimated fuel consumption, grade of air pollution, traffic light status, etc. Such an application is configured according to the particular requirement of the place it is located. If the situation changes it has to adapt itself – taking into account the environ of neighboring roads – by managing traffic lights and applying new strategies, for instance, changing driving directions on multiline highways (when this is possible).

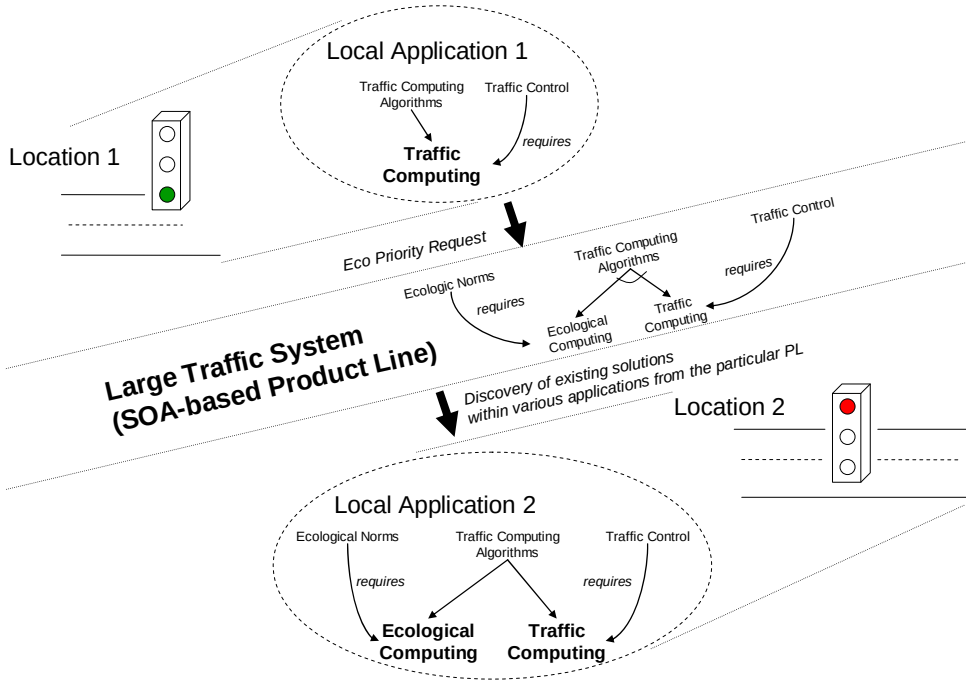

Figure 2. Large Traffic System

The whole large traffic system provides local applications, which are heterogeneous and designed for a particular local use. Thus, the appearance of an unexpected situation which is not covered by an application is not a surprise. Thereby, real time evolution for an application is a must. It can be achieved by using variable mechanism of the infrastructure.

Consider a situation, when a motorway is blocked by an incident, and cars have to take a bypass through a small town, where such a situation was not foreseen. The sudden appearance of a large number of cars is rather critical, thus, there is the necessity to take measures, for instance, to prevent air pollutions by applying strategies not planned beforehand.

The proposed approach assumes the existence of a variability model for the whole large traffic system and a monitoring system, which tracks changes in the behavior of local applications. If, for instance, a device recognizes increased air pollution, it gives a signal whereupon the traffic system changes its signaling patterns: because fuel consumption depends on the number of the cars' stops-and-goes, streets with heavy load will get a prolonged phase of "green light". This re-configuration of the local application relies on selecting appropriate features and tracing their dependencies within the variability tree (see Figure 3).

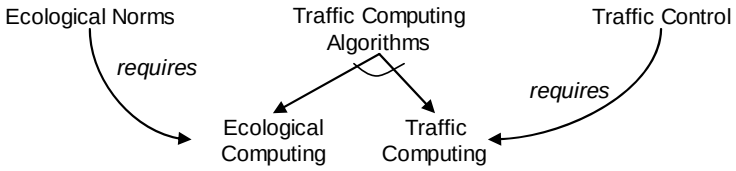

Figure 3. Part of the product line VM "Large Traffic System"

Initially, as it was mentioned above, the actual application uses only traffic-related policies without covering ecological constraints. To cope with the new situation, “Traffic Computing” and “Ecological computing” are combined to satisfy the new requirement “eco priority”.

The presented approach suggests an easy way of identifying an already existing business process from another local application that satisfies the new requirements. It could be part of already existing applications tailored to bigger cities, where considering ecology was already foreseen.

The next step is a combination of the initial process with a new one, where replacement of the activities implies the change in the whole set of required services and infrastructure settings, i.e., installation of the necessary protocols or extending a process with new services.

Conclusion

Adaptive mechanisms for SOA infrastructures are becoming more important with the increasing number of available SOA applications. In fact, without reliable and safe adaptive solutions, it becomes impossible to change existing and to build new services satisfying varying or versatile requirements. The proposed approach together with variability management guarantees that derived composite services with an appropriate infrastructure remain compliant to the varying situations.

Bibliography

[Pohl, 2005] K. Pohl, G. Boeckle, F. v. der Linden. Software Product Line Engineering, Springer, 2005.

[Kakola, 2006] T. Kakola, J.C. Duenas. Software Product Lines, 2006.

[OSGi, 2008] www.osgi.org

[SCA, 2008] www.osoa.org

[Abu-Matar, 2007] Mohammad Abu-Matar Toward a service-oriented analysis and design methodology for software product, 2007 http://www.ibm.com/developerworks/library/ar-soaspl/index.html

Authors' Information

Peter Graubmann – Senior Engineer, CT SE, Siemens AG; e-mail: peter.graubmann@siemens.com

Mikhail Roshchin – PhD Student of Volgograd State Technical University, working in collaboration with CT SE, Siemens AG; e-mail: roshchin@gmail.com