A Model-Driven Approach to Dynamic and Adaptive Service Brokering Using Modes

Howard Foster[1], Arun Mukhija[2],
David S. Rosenblum[2], and Sebastian Uchitel[1]

[1] London Software Systems, Dept. of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK
{hf1,su2}@doc.ic.ac.uk
[2] London Software Systems, Dept. of Computer Science, University College London,
Gower Street, London WC1E 6BT, UK
{a.mukhija,d.rosenblum}@cs.ucl.ac.uk

Abstract. Industry and academia are exploring ways to exploit the services paradigm to assist in the challenges of software self-management. In this paper we present a novel approach which aims to bring these two fields closer by specifying the requirements and capabilities within a UML2 model architecture style and illustrating how these model elements are used to generate specifications for dynamic runtime service brokering given different modes of a software system. The approach is implemented in a tool suite integrated into the Eclipse IDE with a prototype runtime service broker engine.

1 Introduction

Software architectures have typically been specified for a static configuration, where static means that the initial configuration defines a single relationship model between various components in the architecture. With the use of a service-oriented architecture (SOA) style for loosely-coupled reusable software components (typically by technology independence) there is an interest to dynamically configure relationships between these components such that the architecture configuration changes as the system requirements or environment changes (reconfiguration). We introduced the notion of service modes in service-oriented computing in [2] which outlined an approach to defining, abstracting and generating deployment artifacts from component models specified using the mode style. Additionally, dynamic reconfiguration of service architectures also requires that a series of services representing the same functional (or non-functional) requirements may be dynamically composed in to the network of services. As a part of the Dino project [9], we are working on addressing a number of challenges faced by the dynamic and adaptive composition of services in open dynamic environments. Our collective aim in this project is to provide technologies, tools and runtime systems for comprehensively supporting all stages of service engineering (i.e. requirements, discovery, selection, binding, delivery, monitoring and adaptation). In this paper we describe an integrated approach to dynamic service compositions and architecture reconfigurations.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 558–564, 2008.

2 Background and Related Work

A mode, in the context of Service-Oriented Computing (SoC), abstracts a set of services that collaborate towards a common goal [5]. A mode can be used to identify which services are required in states of a system, and assist in specifying service composition requirements through component state changes. Modes can specifically be used towards addressing reconfiguration issues within a self-managed system. Self-management is typically described as a combination of self-assembly, self-healing and self-optimisation. Self-management of systems is not a new idea, with ideas from both the cybernetics and system theory worlds. However in SoC specifically, a dynamic service brokering solution needs to address issues like how to specify the Quality-of-Service (QoS) requirements and capability, and how to select the most appropriate service provider among several functionally-equivalent providers based on the QoS offered. An example service broker engine is called Dino [9]. Dino provides a runtime infrastructure consisting of a number of brokers. These brokers are responsible for, among other things, service discovery and selection on behalf of service requesters. Integrating service modelling, self-management concepts and dynamic service brokering aims at enhancing service engineering to cater for change, adaptive and extendible service solutions.

Related work is split between the modelling and brokering aspects of our work. For modelling requirements of services and SOA there has been several UML profiles proposed in [6,1,7,8]. These profiles generally provide a set of stereotypes that represent features of service artifacts, including a service specification (interface), gateway (ports) and orchestrated collaboration (behaviour specifications). What is generally missing from these existing profile approaches is the ability to identify the requirements and capabilities of services and then to elaborate on the dynamic changes anticipated for self-management. In terms of dynamic service brokering, most of the work on runtime infrastructure for service composition assumes that the global view of an abstract service composition is available centrally, such as the work by Yu et al. [10] and Zeng et al. [11]. In our approach, we do not make such assumptions, and instead allow decentralized composition of services, making use of extended modelling profiles to capture the dynamic change requirements anticipated for self-management.

3 Overview of Approach and Case Study

Our approach is illustrated in Figure 1. Firstly, service engineers create a set of architecture models in UML2 using a Modes Profile to stereotype particular elements needed for service brokering requirements. These models are then used in a second step, to extract the mode related elements from the model and generate inputs to service brokers (in the form of requirements and capability documents). The inputs are passed to a service broker to prepare the required services for matching (both functionally and non-function aspects). Additionally, service provider capabilities can be registered with the broker to announce new

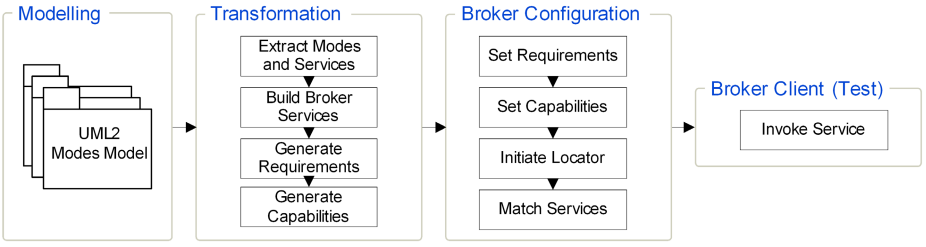

Fig. 1. Approach to Model-Driven Dynamic Service Brokering Requirements with Modes

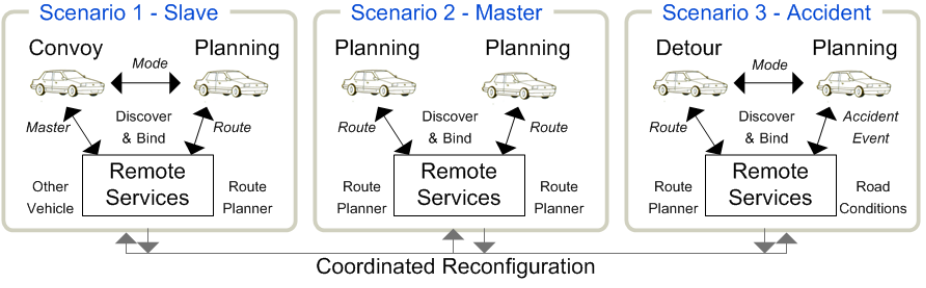

Fig. 2. Mode Scenarios for Vehicle and Remote Service Collaboration

service offerings. Lastly, the broker can be invoked with service parameters to execute a given service call using a matched service linked with the broker.

To guide our work we used a case study based upon a set of requirements formed from those reported as part of a European Union project called SENSORIA [7], our role is to support the deployment and re-engineering aspects of this case study and in particular, to provide a self-management approach. In this case study are a number of scenarios relating to a Vehicle Services Platform and the interactions, events and constraints that are posed on this services architecture. One particular scenario focuses upon Driving Assistance (illustrated in Figure 2), and a navigation system which undertakes route planning and user-interface assistance to a vehicle driver. Modes are used to describe the changes between various states of the vehicle services system.

4 Capturing the Architecture Models and Modes

Using the modelling concepts of UML2 and Modes, we created a UML2 Modes Profile which assists Service Engineers in identifying mode collaborations, organised hierarchically using Mode Packages. Note that this profile extends and depends on the SENSORIA UML for SoC profile [7], for general service stereotypes (such as service, provider, requester etc). The stereotypes for the UML2 Modes Profile are defined in [3], however, here we provide a brief overview. In

this paper we concentrate on architecture, configuration and dynamic service brokering requirements, and as such, examples of behaviour specification and transformation is left as future work.

UML Models and ModePackage. A UML2 model is a package representing a (hierarchical) set of elements that together describe the physical system being modeled. We define a ModePackage stereotype as an extension of this definition with a stereotype to designate that a package is being described for a system mode configuration, events and signals.

ModeCollaboration and ModeBinding. A ModeCollaboration extends a UML2 Collaboration containing a Composite Structure Diagram (CSD) to represent the collaborating service components relationships in this mode and one or more ModeInteraction or ModeActivity diagrams. Within each ModeCollaboration CSD, the UML2 element of Connector is stereotyped as a ModeBinding. A ModeBinding represents a required connection (or instantiation) in order to carry out the mode behaviour as described in a collaboration interaction set.

ModeInteraction and ModeActivity. A ModeInteraction contains a single interaction (sequence) diagram and optionally a single communication diagram. The sequence diagram is a message sequence chart describing the sequence of interactions between service components in the mode collaboration. A communication diagram provides an alternative view of the sequence logic for the mode interactions, in a sense a "bird's eye" view of the way the service components collaborate for a given configuration.

ModeConstraint. Constraining changes to a Modes-based architecture and service composition can be achieved in two ways. Firstly, in a ModeCollaboration specification, a ModeBinding can be constrained with a ModeConstraint, categorised by a further constraint stereotype. We extend a recommendation profile

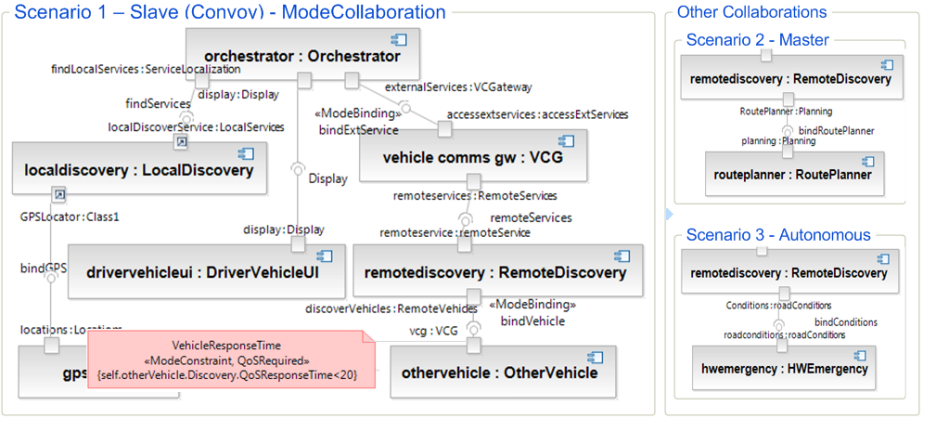

Fig. 3. Composite Structure Diagram (ModeCollaboration) for Slave (Convoy) Component Configuration and alternatives for Planning and Detour Modes

of the Object Management Group (OMG) in [4]. Additionally, architectural constraints may be specified in the Object Constraint Language (OCL) or another constraint based language. The constraint language adopted becomes an implementation-dependent aspect of analysing or extracting from models in UML2. An example constraint for service ResponseTime, applied to a ModeCollaboration, is illustrated in Figure 3.

5 Requirements and Capabilities Specification for Dynamic Brokering

Dino provides a specification language for describing both functional and non-functional properties of service requirements and capabilities [9]. The specification language provided by Dino builds on the advances already made in the field of *semantic* specifications of services. Some of the popular efforts in this direction include, OWL-S, SA-WSDL, and WSMO. The main purpose of the Dino specification language is to provide the *mode* information for different service requirements and capabilities. For practical reasons, we have chosen to specify functional service descriptions using OWL-S in the current Dino broker implementation. This is because OWL-S is a mature standard for semantic service specifications, and a number of tools supporting this standard are available. For specifying the non-functional properties of services, none of the existing semantic service standards were found to be completely satisfactory. Therefore, we have developed our own specification language for describing non-functional properties of services, which relies on a standard QoS ontology. Examples of

```
<ReqDoc name="Driving-Assistant">
   <mode name="planning">
      <service name="GPS" functional="gps-req.owl" qos="gps-req.qos"/>
      <service name="Map" functional="map-req.owl" qos="map-req.qos"/>
   </mode>
   <mode name="convoy">
      <service name="RPS" functional="rps-req.owl" qos="rps-req.qos"/>
   </mode>
   <mode name="detour">
      <service name="HES" functional="hes-req.owl" qos="hes-req.qos"/>
   </mode>
</ReqDoc>

<CapDoc name="Driving-Assistant">
   <mode name="planning, convoy, detour">
      <service name="RPS" functional="rps-cap.owl" qos="rps-cap.qos"/>
   </mode>
</CapDoc>
```

Fig. 4. Requirements (ReqDoc) and Capabilities (CapDoc) of Driving Assistance Modes

non-functional properties include response time, availability, security etc. Details of the specification language for describing non-functional properties in Dino can be found in a previous paper [9]. Figure 4 shows an example requirements specification document for the Driving Assistance case study. As discussed earlier, Driving Assistance has three possible modes: planning, convoy and detour. Driving Assistance requires different services in all three different modes. However, the service provided by Driving Assistance in all three different modes remains the same, i.e. the route planning service, also shown in Figure 4.

A Dino broker can be hosted on a trusted third party node, or even on the same node as a service requester. Any number of Dino brokers can be deployed in an environment, as required for scalability. The brokers perform three main scenarios. Firstly, the service requester can invoke the Dino broker at runtime and forward its requirements specification document to the broker. Secondly, the Dino broker can discover candidate services and perform matchmaking based upon the requirements and thirdly, a service can be delivered either directly from the service provider, or through the Dino broker.

5.1 Extracting Service Requirements and Capabilities

To refine the specification for extracting service requirements and capabilities, we have three main element lists: *a set of mode packages*, *a set of mode collaborations* and *a set of mode components* (services). As described earlier in this section, the Dino broker considers two types of service requirements; 1) required services for a given mode and 2) provided services to announce capabilities that can be matched for a mode of operation. Identifying the type of a service in the mode model relies on alternative properties. The type, requester or provider, can be determined either by direct or profiled stereotype on a given component. The usage of a particular service is analysed by considering the ports and connectors of the component. Identifying the type and usage allows us to generate functional and non-functional broker inputs.

Functional Broker Inputs. If the service component has a provided interface, then each operation type, id and name is appended to the Dino Service representation as provided operations. For a provider of operations, building the Dino Service operations is relatively straightforward. However for a requester type service, the connector between service requester and provider instances must be referenced.

Non-Functional Broker Inputs. We also support extracting ModeConstraints for service bindings, or more specifically a quality of service attribute applied to assembly connectors between two or more components. A ModeConstraint is expected to be expressed in a particular format. The QoSRequired constraint consists of two key aspects. Firstly, that it has applied the QoSRequired stereotype from the QoS Profile, and secondly that it specifies an OCL statement which constraints the binding operation. We have developed a prototype tool suite to mechanically support the approach steps described in this paper as part of our service engineering tool suite, known as WS-Engineer, which is available from **http://www.doc.ic.ac.uk/ltsa/wsengineer**.

6 Conclusions and Future Work

We believe that the notion of Service Modes helps service engineers abstract appropriate elements, behaviour and policy from the services domain, and can facilitate the specification of appropriate control over both architectural change and service behaviour. In this paper we have presented our approach to the modelling of service-oriented computing component architectures using an abstraction of modes to represent the changes in such an architecture. Using a UML2 Modes Profile allowed us to define different capabilities and requirements for dynamic service brokering. Our future work will explore how mode configurations and their constraints are analysed for completeness and correctness. We are also seeking to implement the generation of service orchestrations and choreography from service mode behaviour specifications. This work has been partially sponsored by the EU funded project SENSORIA (IST-2005-016004). David Rosenblum holds a Wolfson Research Merit Award from the Royal Society.

References

1. Ermagan, V., Krüger, I.H.: A uml2 profile for service modeling. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 360–374. Springer, Heidelberg (2007)
2. Foster, H., Uchitel, S., Kramer, J., Magee, J.: Towards self-management in service-oriented computing with modes. In: Workshop on Engineering Service-Oriented Applications, Vienna, Austria (2007)
3. Foster, H., Uchitel, S., Kramer, J., Magee, J.: Leveraging Modes and UML2 for Service Brokering Specifications. In: 4th Model-Driven Web Engineering Workshop (MDWE), Toulouse, France (2008)
4. Object Management Group. Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms. Proposal-AD/02-01/07 (2002)
5. Hirsch, D., Kramer, J., Magee, J., Uchitel, S.: Modes for software architectures. In: Third European Workshop on Software Architecture. Springer, Heidelberg (2006)
6. Johnston, S.: Uml 2.0 profile for software services (2005), `http://www-128.ibm.com/developerworks/rational/library/05/419soa`
7. Koch, N., Mayer, P., Heckel, R., Gonczy, L., Montangero, C.: D1.4b: Uml for service-oriented systems. Technical report (October 2007)
8. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of uml models for service-oriented software architectures. In: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Washington, DC, USA, pp. 173–182 (2005)
9. Mukhija, A., Dingwall-Smith, A., Rosenblum, D.S.: QoS-aware service composition in Dino. In: Proceedings of the 5th IEEE European Conference on Web Services (ECOWS 2007) (November 2007)
10. Yu, T., Lin, K.-J.: A broker-based framework for QoS-aware web service composition. In: Proceedings of the International Conference on e-Technology, e-Commerce and e-Service (EEE 2005) (March-April 2005)
11. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)