Chapter 1
Policy–Based Service Composition and Recommendation

Rolv Bråk  
Norwegian University of Science and Technology, Norway

Humberto Nicolás Castejón  
Telenor GBD&R, Norway

Hien Nam Le  
Norwegian University of Science and Technology, Norway

Judith E. Y. Rossebo  
ABB Corporate Research, Norway

ABSTRACT

This chapter addresses concepts and methods to support dynamic composition of situated services. We focus mainly on service modelling and service design for execution environments that can support dynamic composition of situated services. In our approach, services are modelled using UML 2.x collaborations that are mapped to parts of a UML 2.x design model. Services are also associated with situations, that is, sets of properties that characterise the executing environment of the service. A policy-driven mechanism is proposed to enhance the service composition process. The policy model takes into account context situations and user preferences that can impact the performance and functionalities of the composed services. Within a given situation, executable services are identified and service composition policies used to determine their execution order. We demonstrate the approach using a multi-media over IP service that takes into account security requirements, monitored threat levels, user locations and preferences.

INTRODUCTION

Despite the considerable attention that services and service engineering have received in recent years, one has not yet arrived at a universal understanding of what a service is and what service composition and context adaptation entails. By “universal” understanding we mean a commonly agreed understanding that is not tied to particu-
lar technologies such as web-services, protocol services or middleware services, but allows these as special cases within the wider domain of ICT systems.

As a step towards a universal understanding we venture to define "service" as an identified functionality aiming to establish some desired goals/effects among collaborating entities. This understanding is quite general and captures various common uses of the term "service", such as end-user services meaning functionality provided to end-users, protocol services meaning the functionality provided by a protocol layer to the layer above, component services meaning the functionality provided by a component to its environment across an interface. It is this latter understanding of service that is normally used in middleware such as CORBA, in web services and, to some extent, in the SOA paradigm.

Sometimes it is possible to make a distinction between a service provider that creates the effect and a consumer that makes use of it by means of an interface, but this is not always possible. In a two party call service directly established between two end-points, for instance, it is not so obvious who is the provider and who is the consumer. Both parties participate in creating the effect (i.e., the voice connection) and they both benefit from it. The definition above also covers this kind of peer-to-peer service. According to this understanding, services have several important characteristics:

A service is functionality; it is behaviour performed by entities.

A service implies collaboration; it makes no sense to talk about a service unless at least two entities collaborate (e.g. a client and a server).

Service behaviour is crosscutting; it implies coordination of two or more entity behaviours.

Service behaviour is normally partial; it is usually to be composed with other services to achieve complete system functionality.

It is the collaborative, crosscutting and partial nature of services that makes service engineering, in general, such a challenge. Ideally one wants to define services separately so they can be analyzed, understood and reused as separate entities; and then be able to compose them into well-functioning systems that can adapt to different context situations and individual preferences (Sanders, Castejón, et al. 2005; Papazolou, Traverso, et al., 2008). This can be achieved by separating service models from design models and implementations as illustrated in Figure 1.

Service models are used to precisely define and compose services. The new collaboration concept introduced in UML 2.0 (OMG, 2004) provides an excellent basis for service modelling. A collaboration defines a structure of roles (i.e. partial object behaviours) that cooperate with each other to achieve a common task or goal, so it closely corresponds to the concept of service as defined here. Moreover one may compose collaborations from smaller collaborations, representing sub-services, by means of collaboration uses.

Design models define design solutions as structures of interconnected parts having well-defined behaviour. UML structured classes with inner parts and state machines provide a good basis for design models. By binding the roles of service collaborations to parts in the design structure one defines the relationship between service models and design models. In general, a part will be able to participate in several services by playing different roles, and a service will involve roles played by several different parts. This illustrates that services and roles add a dimension of composition (and flexibility) on top of parts and classes.

Realizations are implementations that execute on different service platforms. Using state of the art MDE techniques, they may be automatically derived from service models and design models.

In previous publications (Castejón, Bochmann, et al., 2007; Kraemer, Bræk, et al., 2007) we have described methods and tools that enable precise definition of service models, automatic synthesis of design models from service models and automatic generation of executable implementations from design models, as well as formal analysis of