

Metamodel-Based Refinement of Dynamic Software Architectures

Luciano Baresi  
(Politecnico di Milano)

Reiko Heckel  
(University of Paderborn)

Sebastian Thöne  
(Int. Graduate School Paderborn)

Dániel Varró  
(Budapest University of Technology and Economics)

# **A UML-Profile for Service-Oriented Architectures**

Paderborn, September 2003

Contact: Sebastian Thöne  
Int. Graduate School Dynamic Intelligent Systems  
University of Paderborn  
D-33095 Paderborn, Germany  
[seb@upb.de](mailto:seb@upb.de)  
[www.upb.de/cs/ag-engels/ag\\_engl/People/Thoene/MRDSA/](http://www.upb.de/cs/ag-engels/ag_engl/People/Thoene/MRDSA/)

# 1 Introduction

This document defines a UML-profile as an extension for modelling service-oriented architectures. It adapts the UML metamodel as defined in the specification document [OMG03] by stereotypes for service-oriented architectures.

## 1.1 Status of this document

This is the first version of this document.

## 1.2 Overview Service-Oriented Architectures

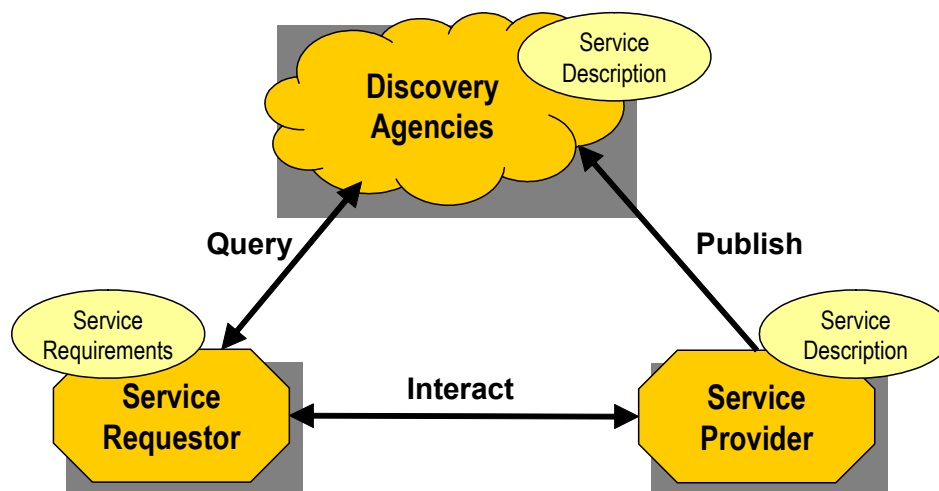


Figure 1: Service-Oriented Architectures

Service-oriented architectures involve three different kinds of actors: *service providers*, *service requesters* and *discovery agencies*. The service provider exposes some software functionality as a service over a network to its clients. In order to allow requesters to access this service, the provider has to publish a *service description*.

Since service provider and service requester usually do not know each other in advance, the service descriptions are published via specialized discovery agencies. They categorize the service descriptions and deliver them in response to queries issued by service requesters. As soon as the service requester has retrieved a service description meeting its *service requirements*, it can use it to interact with the service.

Service-oriented architectures are typically highly dynamic and flexible: Components and services are only loosely coupled and communicate according to standardized protocols; interface specifications are exchanged at run-time and, thus, clients can replace services at run-time. This might be advantageous if a new service provides a better alternative to the former one concerning functionality or quality of service. Or, it might become necessary for self-healing purposes, e.g., if a service is not reachable any longer because of network problems.

## 2 Stereotype Definitions

Stereotypes define how existing metaclasses are extended and enable the use of platform specific elements and notations in the extended models. The following figure defines the stereotypes of the SOA profile including the metaclasses they extend.

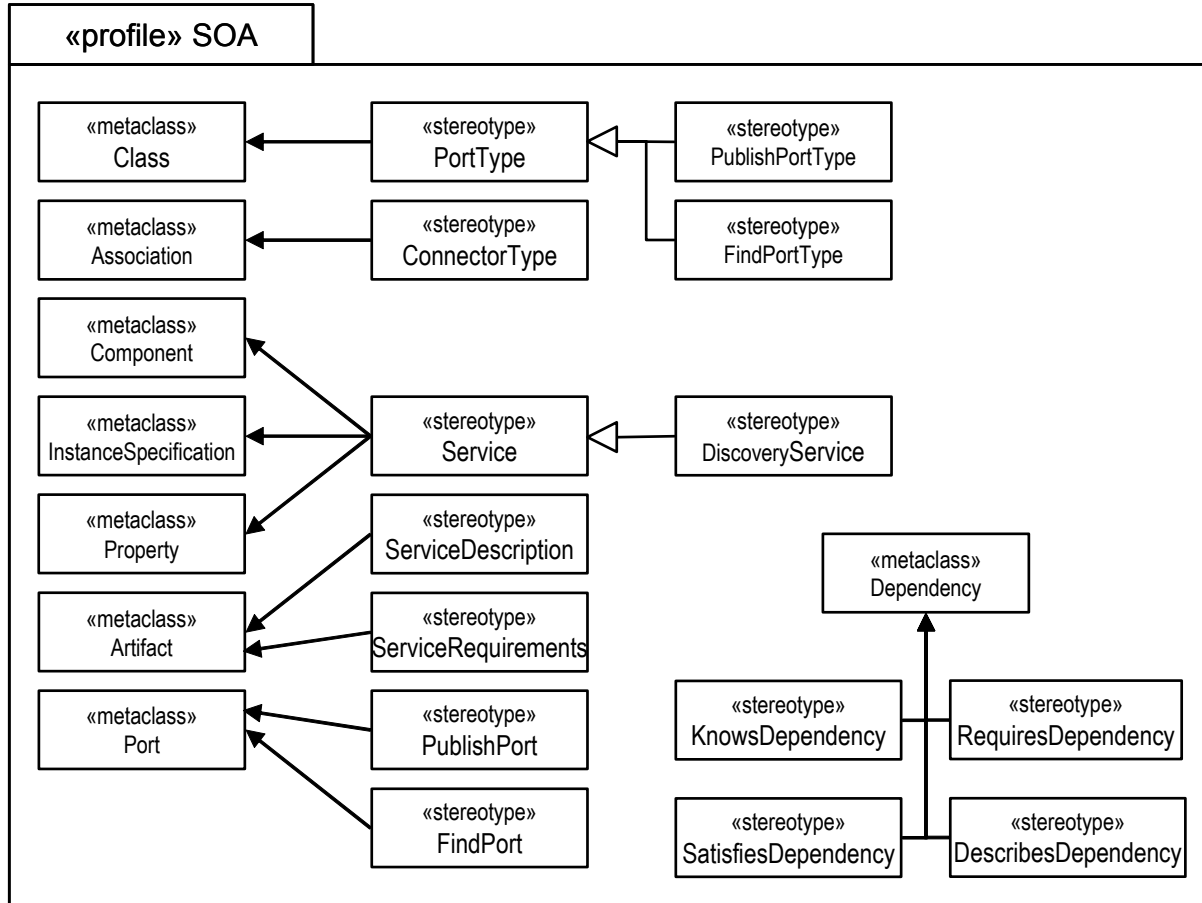


Figure 2: Stereotypes and their base classes

### 3 Stereotype Notations

The following table assigns appropriate notations to the above defined stereotypes. In the default case, the stereotype label is attached to the notation of its base class.





Stereotype from SOA profile	Notation
PortType	«portType»
PublishPortType	«publishPT»
FindPortType	«findPT»
ConnectorType	«connectorType»
Service	«service»
DiscoveryService	«discovery»
ServiceRequirements	
ServiceDescription	
PublishPort	
FindPort	
KnowsDependency	«know»
SatisfiesDependency	«satisfy»
DescribesDependency	«describe»
RequiresDependency	«require»

Figure 3: Stereotypes and their notation

## 4 Relation to the SOA-Architectural Style

The above defined profile can be used to provide a concrete syntax to architectural models that follow the SOA-specific architectural style defined in [BHTV03].

In order to define the relationship between syntax and semantics, the following tables enumerate all elements defined in the type graph of the architectural style and assign a class of the UML 2.0 metamodel plus the UML profile for SOA to each element.

Since UML distinguishes between instance-level and prototypical-level models, while the architectural style does not, there are in some cases two slightly different options for the UML element.

The third column of the tables exemplifies how to depict the chosen UML element.

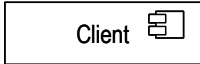





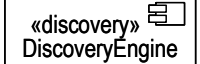
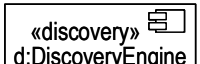
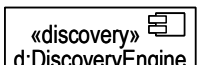
Element of Architectural Style	Notation defined by element of UML 2.0 metamodel plus UML Profile for SOA	Example
Component	BasicComponents::Component	
ComponentInstance	If used on the instance-level: Kernel::InstanceSpecification	
	If used on the prototype-level: StructuredClassifier::Property	
Service	BasicComponents::Component stereotyped by SOA::Service	
ServiceInstance	If used on the instance-level: Kernel::InstanceSpecification stereotyped by SOA::Service	
	If used on the prototype-level: InternalStructures::Prototype stereotyped by SOA::Service	
DiscoveryService	BasicComponents::Component stereotyped by SOA::DiscoveryService	
DiscoveryServiceInstance	If used on the instance-level: Kernel::InstanceSpecification stereotyped by SOA::DiscoveryService	
	If used on the prototype-level: InternalStructures::Prototype stereotyped by SOA::DiscoveryService	

Figure 4: Relating architectural style elements and UML elements (I)


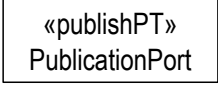
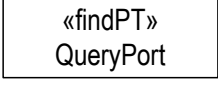



Element of Architectural Style	Notation defined by element of UML 2.0 metamodel plus UML Profile for SOA	Example
PortType	Kernel::Class stereotyped by SOA::PortType	
PublishPort	Kernel::Class stereotyped by SOA::PublishPortType	
FindPort	Kernel::Class stereotyped by SOA::FindPortType	
Port	If self.type.ocllsTypeOf(PortType) Ports::Port	port:PortType 
	If self.type.ocllsTypeOf(PublishPort) Ports::Port stereotyped by SOA::PublishPort	port:PortType 
	If self.type.ocllsTypeOf(FindPort) Ports::Port stereotyped by SOA::FindPort	port:PortType 

Figure 5: Relating architectural style elements and UML elements (II)

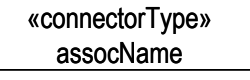

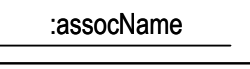
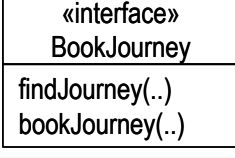


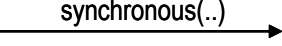
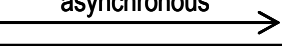
Element of Architectural Style	Notation defined by element of UML 2.0 metamodel plus UML Profile for SOA	Example
ConnectorType	Kernel::Association stereotyped by SOA::ConnectorType	
Connection	If used on the instance-level: Kernel::InstanceSpecification	
	If used on the prototype-level: BasicComponents::Connector	
Interface	Interfaces::Interface	
Operation	Kernel::Operation	
ServiceSpecification	Artifacts::Artifact	
ServiceRequirements	Artifacts::Artifact stereotyped by SOA::ServiceRequirements	
ServiceDescription	Artifacts::Artifact stereotyped by SOA::ServiceDescription	
Message	Messages::Message	
		

Figure 6: Relating architectural style elements and UML elements (III)


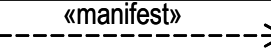
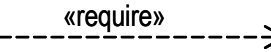
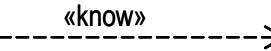

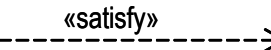
Relationship of Architectural Style	Notation defined by element of UML 2.0 metamodel plus UML Profile for SOA	Example
<i>PortType refines PortType</i>	Kernel::Generalization	
<i>ServiceSpecification manifests Port</i>	Artifacts::Manifestation	
<i>ComponentInstance requiresServiceFor ServiceRequirements</i>	Dependencies::Dependency stereotyped by SOA::RequiresDependency	
<i>ComponentInstance knows ServiceDescription</i>	Dependencies::Dependency stereotyped by SOA::KnowsDependency	
<i>ServiceDescription describes Service</i>	Dependencies::Dependency stereotyped by SOA::DescribesDependency	
<i>ServiceDescription satisfies ServiceRequirements</i>	Dependencies::Dependency stereotyped by SOA::SatisfiesDependency	

Figure 7: Relating architectural style elements and UML elements (IV)

## 5 Example Diagrams

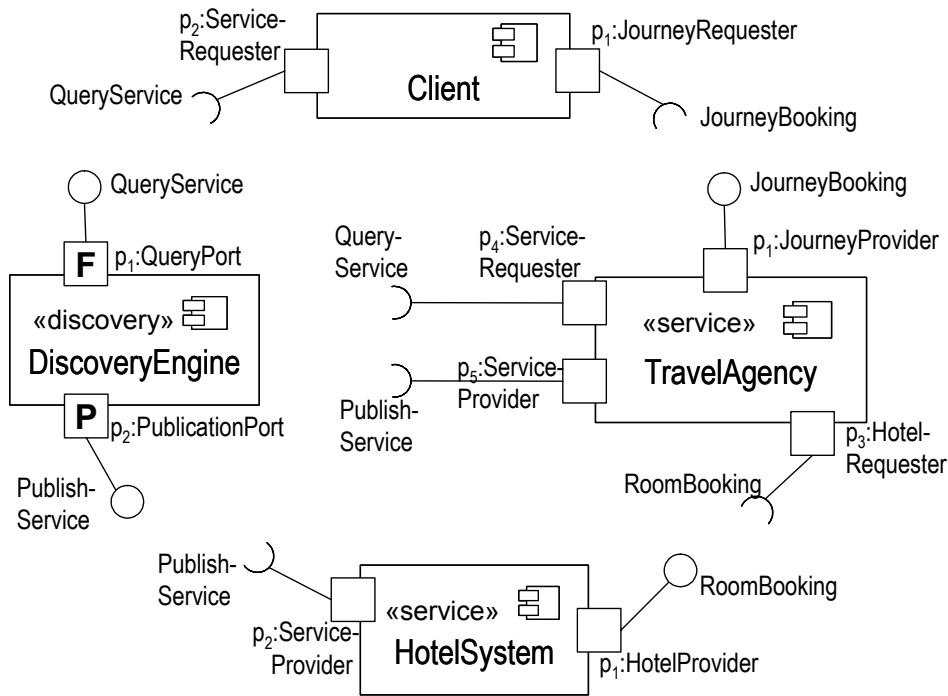
Consider a travel agency application that serves as a manager for planning and booking journeys. For simplicity, we restrict the functionality to booking suitable hotel accommodation.

For this purpose, the travel software should be able to connect to external hotel information systems. After a client's request for a journey has arrived, the system has to query these third-party systems for suitable offers, then chooses the best offer, and books the corresponding hotel.

These requirements lead to a dynamic, distributed system where new components can be brought up by hotel companies at execution-time. Thus, the application is a candidate for being realized as a service oriented architecture, as modelled with the UML profile for SOA in the following diagrams.

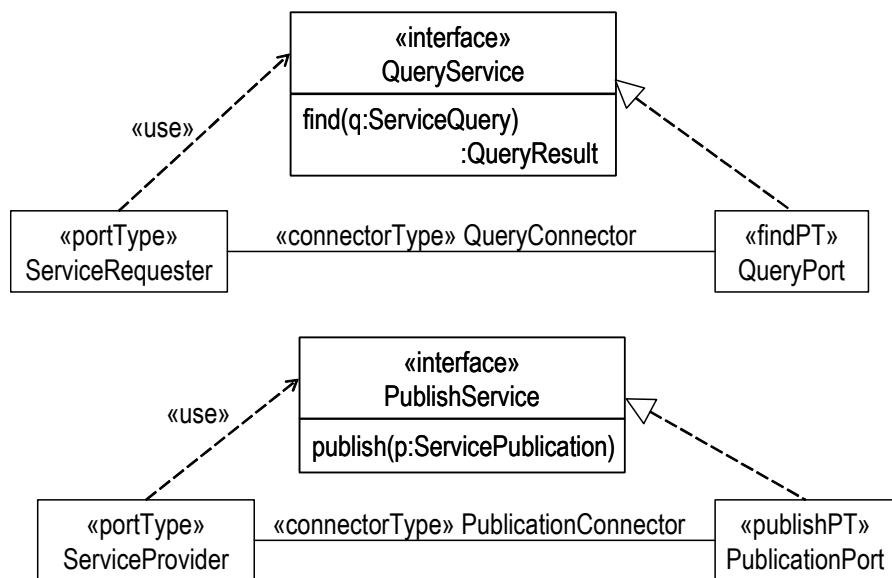
We distinguish between models on the *type* level and on the *instance* or *prototypical* level: At the type level component diagrams are used to show the component types of the application including the interfaces that are provided and required by their ports (see Figure 8).

The Client component requests a journey from the TravelAgency component, which then connects to different HotelSystems. According to SOA (cf. Section 1.2) an additional component playing the role of a discovery service is required.



**Figure 8: SOA-specific component diagram**

The details of the port types and the provided and required interfaces are defined by class diagrams (partially shown in Figure 9). The class diagram also contains stereotyped associations which define types for connectors that are available to connect different port types.



**Figure 9: SOA-specific class diagram**

The instance or prototypical level comprises configurations of component instances or of placeholders, which are bound to concrete instances at execution-time. To model such configurations, we use UML collaboration diagrams like in Figure 10.

If the underlying configuration, modelled as a set of component instances and connections, is to be changed during execution, we assign `{new}`, `{transient}` or `{destroyed}` labels to the



