Sensoria

016004
Software Engineering for Service-Oriented Overlay Computers

D1.4a: UML for Service-Oriented Systems

Lead contractor for deliverable: LMU
Author(s): Nora Koch (FAST & LMU), Philip Mayer (LMU), Reiko Heckel (ULEICES), László Gönczy (BUTE), Carlo Montangero (PISA)

Due date of deliverable: 31.08.2007
Actual submission date: 14.10.2007
Revision: Final
Classification: PU

Contract Start Date: September 1, 2005 Duration: 48 months
Project Coordinator: LMU
Partners: LMU, UNITN, ULEICES, UWARSAW, DTU, PISA, DSIUF, UNIBO, ISTI, FFCUL, UEDIN, ATX, TILab, FAST, BUTE, S&N, LSS-Imperial, LSS-UCL, MIP, ATXT

Integrated Project funded by the European Community under the "Information Society Technologies" Programme (2002—2006)

Executive Summary

This deliverable presents a UML-based modelling approach for service-oriented systems. The aim is to introduce service-specific model elements to ease the modelling activity of service-oriented architectures (SOAs). The UML 2.0 extension is built on top of a Meta Object Facility (MOF) metamodel defined as a conservative extension of the UML metamodel. For the elements of this metamodel, a UML profile is created using the extension mechanisms provided by the UML. The result is a so-called UML profile for a service-oriented architecture (UML4SOA), which provides model elements for structural and behavioural aspects, and for business goals, policies and non-functional properties of SOAs. Such a metamodel and the corresponding UML profile constitute the basis for model transformations and code generation defining a model-driven development process. The distinguishing feature of UML4SOA is its compliance with standards like UML 2.0 , MOF, OCL and XMI.

Contents

1 Introduction

The UML 2.0 is accepted as lingua franca in the development of software systems. To use the UML for modelling service-oriented systems has many advantages when compared to the use of proprietary modelling techniques for a project. These advantages are on the one hand the existing CASE tool support, which is provided by commercial and open source tools. On the other hand, it avoids the definition from scratch of a new modelling language, which would require an own project to detail their syntax, semantics and provide user-friendly tool support. In addition, the UML provides flexible extension mechanisms for defining a domain-specific language (DSL) – a so-called UML profile. Examples of such domain-specific languages are: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms, UML Profile for Schedulability, Performance and Time, UML Profile for System on a Chip (SoC), UML Testing Profile, etc. (see [OMGP])

The objective of a UML profile for service-oriented architectures (SOAs) is to provide a notation that allows for intuitive and expressive specifications of service-oriented systems. Hence, the benefit of such an extension when compared to the use of “pure” UML is to provide model elements for concepts frequently used in the modelling of service-oriented architectures and to define for them a well-defined semantics. Some of these elements provide shortcuts for service patterns.

First, we built a MOF metamodel of service-oriented concepts, which are required to specify SOAs. This metamodel is based on the SENSORIA ontology. In the graphical representation of this metamodel we show some UML metaclasses, those which are directly related to the new SOA concepts. The metamodel is defined as a conservative extension of the UML, i.e. the UML metamodel is not altered. Relationships between SOA elements and UML elements are restricted to inheritance from UML elements and directed associations to UML elements.

For each element of the metamodel a stereotype is defined establishing an extension relationship to an appropriate UML metaclass. Tagged values and OCL constraints are optionally added in order to specify restrictions. Most of the available CASE tools support the use of profiles that can be imported and applied to different modelling projects.

In Section 3 we describe the metamodel on which UML4SOA is based. In Section 4, the UML Profile is presented as an extension of model elements of the UML 2.0. Finally some examples are given in Section 5.

2 UML4SOA: A Domain-Specific UML Extension

The Service-Oriented Architecture (SOA) introduces many new concepts and architectural patterns to the modelling of software systems. In order to allow developers and modellers to use these concepts to their full potential, a modelling language like UML 2.0 needs to be extended with specific elements which are geared towards expressing the new concepts on the right level of abstraction.

To enable developers to model SOA applications in a straightforward way, we have devised the Service-Oriented Profile (UML4SOA), a UML extension which defines a so-called domain-specific language (DSL) for service-oriented systems. In this language, the main concepts introduced by Service-Oriented Architectures are used as shown in Figure 1, which presents an overview of both the structural and behavioural aspects of SOAs.

It introduces the following concepts for the structural view of service-oriented systems:

- **Components** that provide or require services.
- **Services** are provided by components, which implement services as ports. Attached to a service are provided and required interfaces.
- **Provided interfaces** contain operations that are implemented by the service itself.
- **Required interfaces** contain operations, which the service needs, i.e. they must be implemented by other parties (partners), which are then bound to the service.

In general, SOAs are used for modelling and implementing complex business applications. Therefore, we need to cover not only the structural and behavioural view of the SOA system, but also

- **business goals** and **policies;**
- **non-functional properties** of services.

These can be classified as crosscutting concerns and potentially affect all of the elements introduced above.

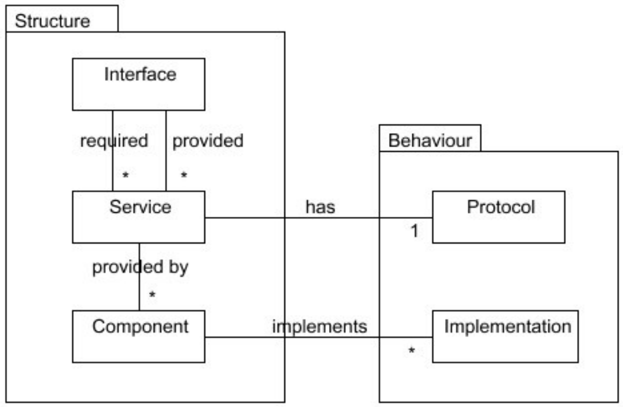

Figure 1: Main concepts in the Service-Oriented Architecture Profile (UML4SOA)

Service-Oriented Architectures place great emphasis on the composition, or orchestration, of services. In our profile, we introduce specific support for implementing services as compositions by providing elements for modelling a complete behavioural specification of a composed service.

2.1 Modelling Structure

The metamodel includes model elements to support specification of structural aspects of services. The UML 2.0 extension comprises model elements for UML structure diagrams (see Figure 2) and UML deployment diagrams, which are shown in Figure 3. Service, service interface and service description as well as service provider and service requester are concepts that were defined in the Ontology, which is a result of *Task 1.1* in the SENSORIA project.

In all our diagrams, we represent existing classes from the UML metamodel with a yellow background, while the new UML4SOA classes have a white background.

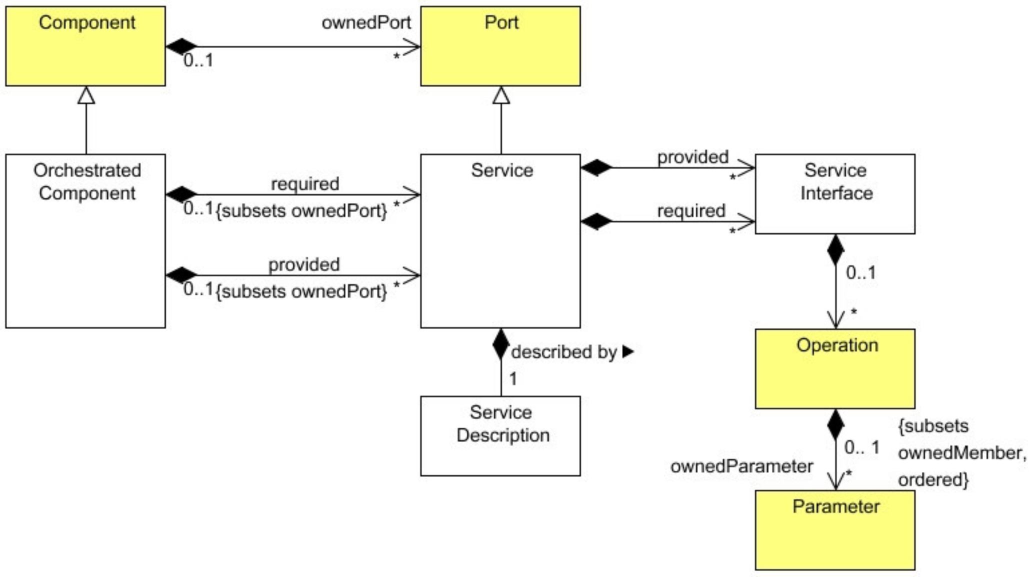

Figure 2: UML4SOA metamodel: package Structure

The following important relationships are defined in this part of the model:

- An **orchestrated component** may contain several services implemented as ports. The orchestration of these services define a new service.
- Each service may contain **a required and a provided interface**.
- Each interface may contain **operations**. These operations, in turn, may contain an arbitrary number of **parameters**.
- A service is described by a **service description.**

Figure 3(a) shows both the UML model elements (yellow background) and Figure 3(b) the UML4SOA concepts defined for permanent, temporary and on-the-fly communication paths, which are used in modelling architectures by deployment diagrams. These temporary and on-the-fly communication paths are typical for SOA architectures.

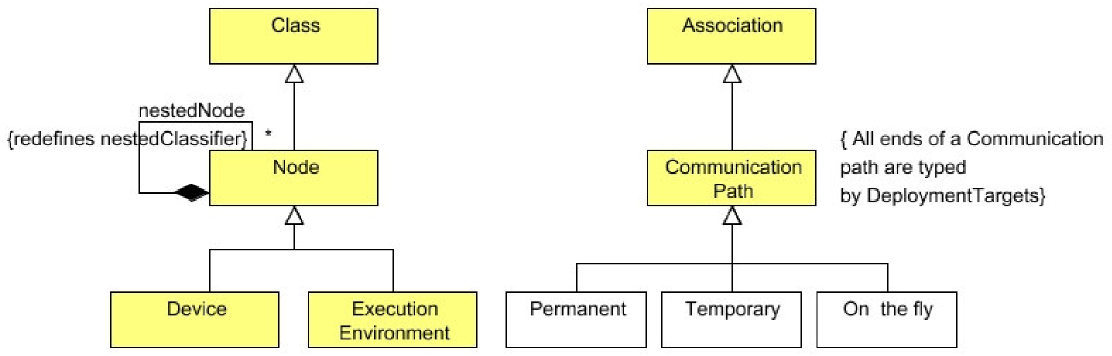

Figure 3: Deployment: (a) UML model elements (b) UML4SOA metamodel: package Structure

For each element of the structural package of the metamodel a stereotype for the Service-Oriented Profile (UML4SOA) is defined (see section 3). A description of each element is given in that section as well.

2.2 Modelling Behaviour

Our following metamodel includes UML4SOA model elements that support the specification of behavioural aspects of service-oriented systems, which comprise the implementation of the components and the service behaviour described by its protocol. The focus is on service interactions, long running transactions and their compensation and exception handling. The metamodel is presented in two parts: In the next section, service orchestration along with compensation activities is shown. In Section 2.2.2 and 2.2.3 we briefly discuss modelling of the service protocol and requirements for SOAs .

2.2.1 Modelling Service Orchestration

Service orchestration is the process of combining existing services together to form a new service to be used like any other service. To allow modelling of such compositions in UML, we add specific service-aware elements to be used in activity diagrams.

The metamodel depicted in Figure 4 details the definition of orchestration activities and the information sent and received by messages. The following relationships are defined in this figure:

- An orchestrated component contains an **orchestration** as its implementation.
- The orchestration contains a **root scope**, which in turn contains all necessary elements for the workflow modelling.
- The **service interaction** has been defined specifically for service interaction, like sending and receiving data. It is defined as an UML abstract class.
- Service interactions may have **interaction pins** for sending or receiving data.

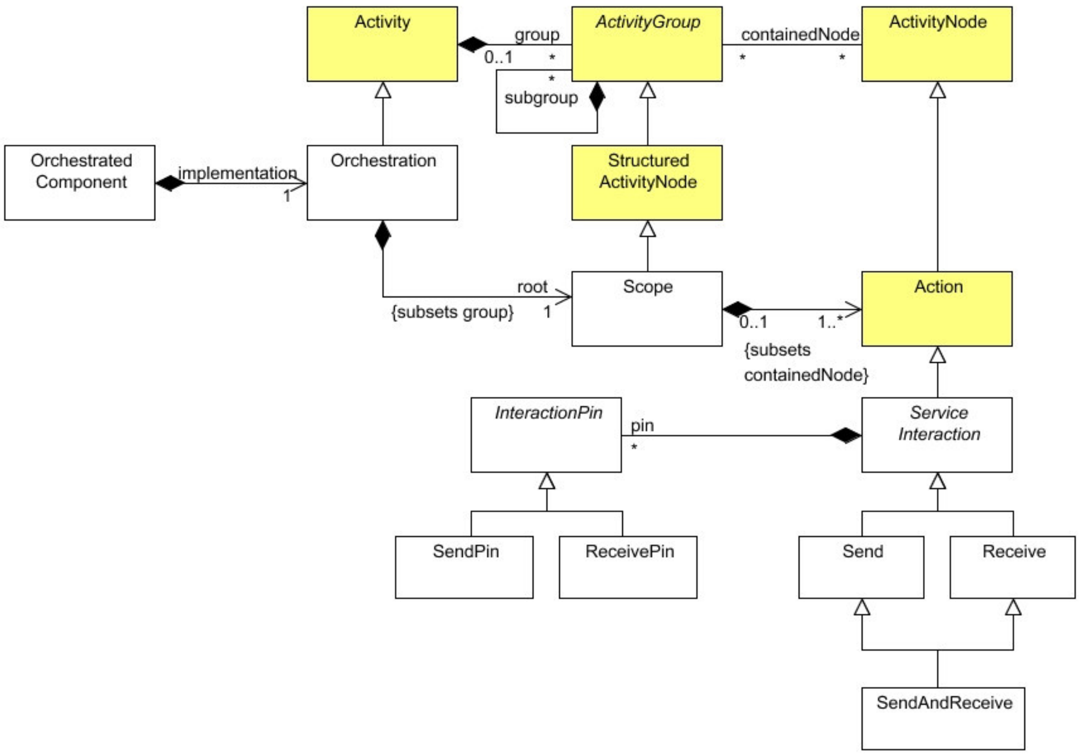

Figure 4: UML4SOA metamodel: package Behaviour - Orchestration

In addition, the metamodel shown in Figure 5 presents new elements to be used for compensation activities. In particular, it defines:

- Compensation edges which link orchestration activies to other activities that model the compensation.
- Compensate and CompensateAll activities which can be used to compensate scopes.

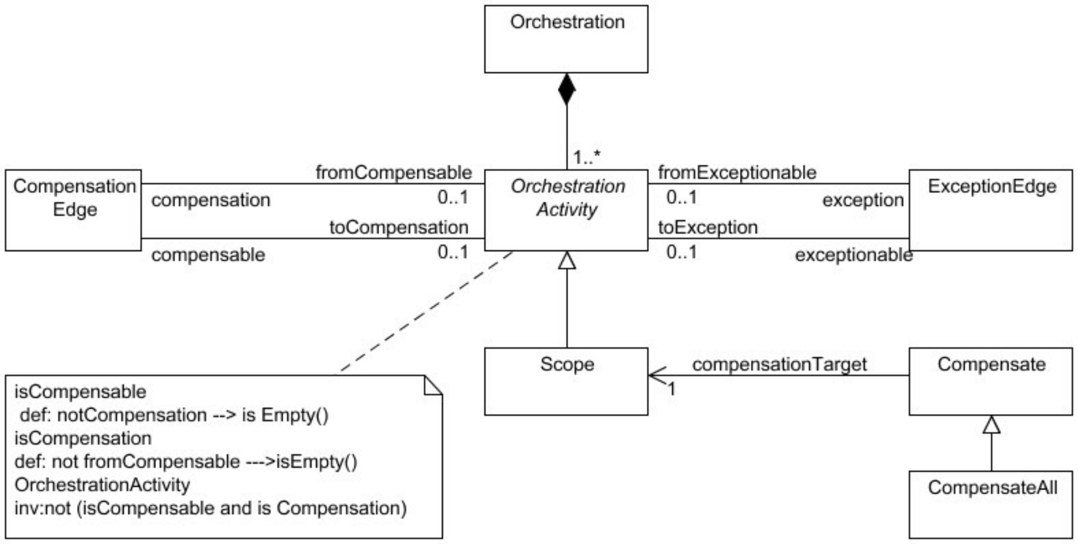

Figure 5: UML4SOA metamodel: package Behaviour - Compensation

2.2.2 Modelling Service Protocol

At service level, the behaviour of the port providing services to or requiring services from other parties is specified using an UML state machine. Future work will be a detailed analysis of the requirements for these state machines for service-oriented systems. The requirements are mainly given by the transformations to be defined to follow a model-driven approach. See Figure 6 for a general approach based on general UML protocol state machines.

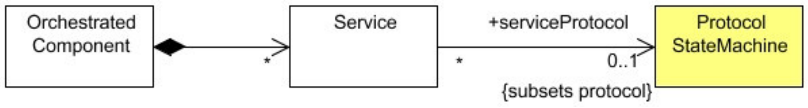

Figure 6: UML4SOA metamodel: package Behaviour - Protocol

2.2.3 Modelling Requirements for SOAs

The UML4SOA approach provides in addition a way to specify at requirements specification level which functionality of a SOA will be implemented and published as services, i.e. a specific service-oriented use case is defined.

For each element of the behavioural package of the metamodel a stereotype for the Service-Oriented Profile (UML4SOA) is defined (see Section 3). A description of each element is given in that section as well.

2.3 Modelling Business Goals and Policies

This section discusses how UML can support business process modelling, and especially its variability via policies, as expressed in the Service-targeted Policy-oriented Workflow Language (StPowla) [GMRFS07]. StPowla introduces a novel combination of policies and workflows that adds to each of the concepts being used on their own, since it permits to capture the essential requirements of a business process by workflow notation, and at the same time to express the inevitable requirements variability by policies in a descriptive way, at a similar level of abstraction. Besides this contribution to the SENSORIA software development process, StPowla makes a contribution to the engineering of service-oriented systems, since it creates a clear link between this enhanced workflow mechanism and services. Tasks are being executed by services, and StPowla permits to define requirements and SLAs that together specify which service can be chosen and what guarantees it has to provide.

We show how there is a limited need for extensions to UML to capture the needed modelling concepts, provided that the business models are built in the context of a suitable framework. The StPowla profile introduces stereotypes, to characterize the tasks in a workflow linking them to services and policies. The StPowla framework has three components: the workflow notation, which is used to express the core flow of the business; the policy language, which is used to adapt the workflow to the varied requirements of the stakeholders; and the specialization of the policy language to implement workflows via services.

The role that UML can play is different in the three components. Representing workflows has been done graphically since the early time of business process modelling, and so naturally lends itself to a representation in UML. Also, activity diagrams are natural candidates for workflow representation. The business process core is defined in terms of sequential, parallel and decision-based composition of building blocks called *tasks*. Then we need to capture in UML the semantics of StPowla tasks. In StPowla, a task has a *type*, which identifies its functionality in business terms, as well as *attributes.* These are used to refer to the state of the execution of the workflow, and characterise properties of individual tasks or of the whole workflow. Besides, a task may have input and output values. The main point in StPowla is that the task is carried out by (at least) one service, and that the choice of the service(s) may be specified by *policies* associated to the task. A *default policy* is initially associated with each task: when the control reaches the task, a service is looked for, bound and invoked, to perform the main functionality of the task, as expressed by the constraint.

The policies can express their choices by inspecting and/or using both the attributes and the input values. The way policies constrain the choice of the implementing service(s) is by specifying the service SLAs, according to predefined *dimensions* that define the admissible variability. For instance, in the example in Section 4.4 we use the dimension `Automation`, which may take two values, i.e. `interactive` and `automated`.

The behaviour of tasks is represented by the state machine of Figure 11 (where we are loose with respect to the arguments and the results of the task/service). It is assumed that there are two engines at run-time, which exchange signals and take care of the execution of workflows and policies, respectively. Then, upon activation of the task, the *WorkflowEngine* signals the *PolicyEngine* that the task has been entered, and then waits for the outcome of the policy from the *PolicyEngine.* If the policy succeeds, the results are made available to the *WorkflowEngine* to proceed with the next task. If the policy fails, any relevant information is returned via an exception, which may be captured at the workflow level, e.g. in a choice pattern [GRFT07].

Finally, StPowla is targeted to SOA. Its users, though informatically naives, should be aware that the business is ultimately carried out by ***services***, i.e. computational entities that are characterized by two series of parameters: the invocation parameters (related to their functionalities), and the ***Service Level Agreement (SLA)*** parameters. Stakeholders can adapt the core workflow by modifying these agreements. Tasks are the units where BPM and SOA converge, and where adaptation occurs, by using policies: the intuitive notion of task is revisited to offer the novel combination of services and policies offered in Figure 7.

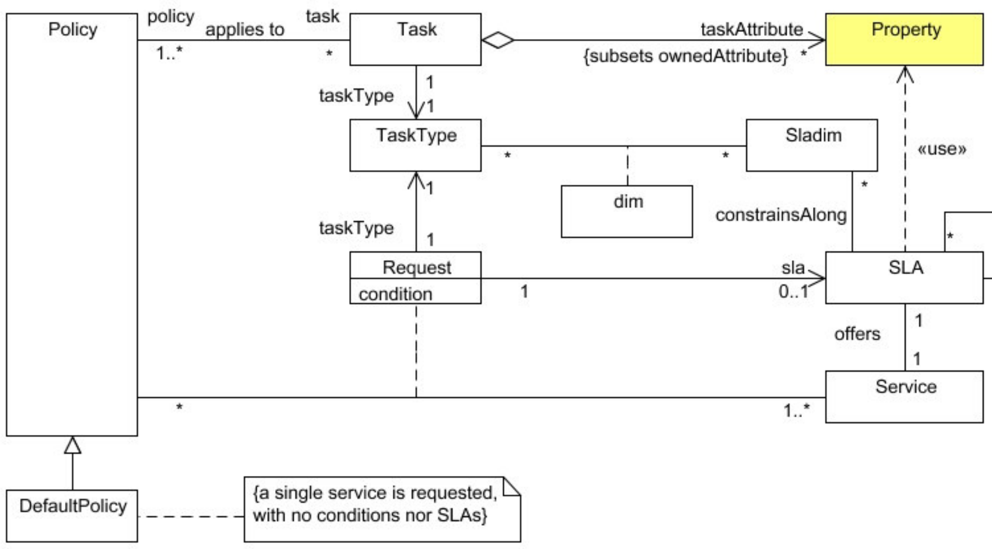

Figure 7: Metamodel for tasks and policies

In general a policy may find several services that satisfy several requests, and bind and invoke them. Each request has a *condition*, which restricts its application to some states of the workflow, identifies a service signature (*TaskType*), and may have SLAs that may involve attributes of the workflow and tasks. There is a notion of match between the offered and requested SLAs. A possible implementation of these requirements, which uses the policy language Appel following [GMRFS07], is exemplified below, in Section 4.4.

2.4 Modelling Non-functional Properties of Services

This section describes the UML extension for modelling non-functional parameters of services. The semantical basis for this extension is defined in the SENSORIA Ontology in View 3 (Non-functional properties of services) while the style of the UML metamodel and the refinement of the concepts is inspired by the ”UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms”. Relevant part of General Resource Model profile was also considered.

Our profile is a UML implementation of the SENSORIA Ontology, View Non-functional properties of services [BFGK06], refined to meet the needs of service development. The style and elements of the UML metamodel is guided by the OMG specification UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [Obj06]. The UML Profile for Schedulability, Performance and Time [Obj05] is a previous related specification of OMG, targeting mainly the modelling issues of realtime and embedded systems. Web Services Service Agreement Language (WSLA, [IBM] by IBM) is another related industrial specification, dealing with the detailed definition of SLAs in the SOA context. Service Component

Architecture (SCA) is a specification for composing services implemented in heterogeneous technologies (Web services, Java, PHP, etc.).

SCA consortium in [Con07] proposes to use descriptions in WS-Policy language to describe non-functional extensions (called intents). Intents are translated to policies depending on the capabilities and standards of the particular implementation platform. However, currently there is no visual modelling support for this mechanism. [OH] presents a profile for extra-functional properties of Web services using Aspect Oriented Programming as implementation technique and WS-Policy as service descriptor. In the SENSORIA project, [GV06] shows an example for an extension of the core SOA model using Reliable Messaging as a case study.

As described in the Technical Annex of SENSORIA, modelling concepts are based on a common knowledge, represented by the SENSORIA Ontology [BFGK06]. This document defines a non-exclusive set of elements to describe the non-functional properties of a service. On the other hand, as already mentioned above, we adapted the QoS and FT extension of OMG to the needs of SENSORIA.

Figure 8 shows the metamodel of the extension. Notes refer to relationship with elements of SENSORIA Ontology. In this extension, we focused on the Quality of Service parameters. The concept of ”QoS” refers to any kind of runtime attribute which can be measured (in terms of qualitative or quantitative metrics).

Using an SLA-driven approach, the profile is built to support contracts (*NFContract,* agreement in the SENSORIA ontology) between provider and requester parties, optionally using external third-party services for monitoring runtime characteristics. These are represented by *requester* and *provider* roles and a *Monitor* class (see SENSORIA ontology [BFGK06]), respectively. A certain non-functional aspect of service (e.g., Performance) is defined as a *NFCharacteristic* (defined as property in the SENSORIA ontology). These characteristics are described by a set of attributes (e.g., Throughput) called *NFDimensions* (defined as attributes in the SENSORIA ontology).

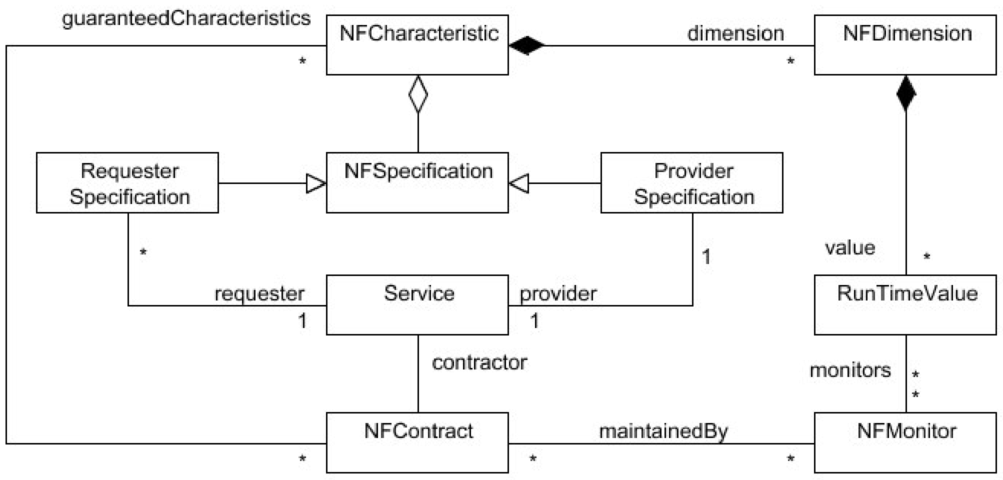

Figure 8: UML4SOA metamodel: non-functional properties

3 A UML Profile for Service-Oriented Architecture (UML4SOA)

In order to be able to use the elements of the UML4SOA metamodel in UML 2.0 CASE tools that are extensible with UML profiles, the profile must be specified by means of stereotypes and their relationships to the classes of the UML metamodel.

Our profile will be presented in four parts. The first subsection shows extensions for modelling the structural specification. The second one includes the stereotypes for the behavioural specification. The third and fourth subsections address stereotypes defined for modelling business goals, policies, and non-functional properties of services. We define a stereotype for all non-abstract classes of the metamodel defined in the previous section (excluding UML classes). The objective is to have a distinct graphical representation and clear semantics for service-oriented concepts.

3.1 UML Extension for Structural Modelling of Services

The UML profile for service-oriented systems includes the following constructs for modelling the structure of SOAs: A stereotype *OrchestratedComponent* indicating that the component is the result of the orchestration of services, stereotypes for specifying services, i.e. *Service, ServiceInterface,* and *ServiceDescription* and stereotypes for distinguishing different types of communication paths, i.e. *Permanent, Temporary* and the more service specific *OnTheFly*.

Figure 9 depicts a UML structure diagrams showing these elements of the UML4SOA profile and the corresponding UML elements, for which they define an extension. Table 1 provide a description of each stereotype.

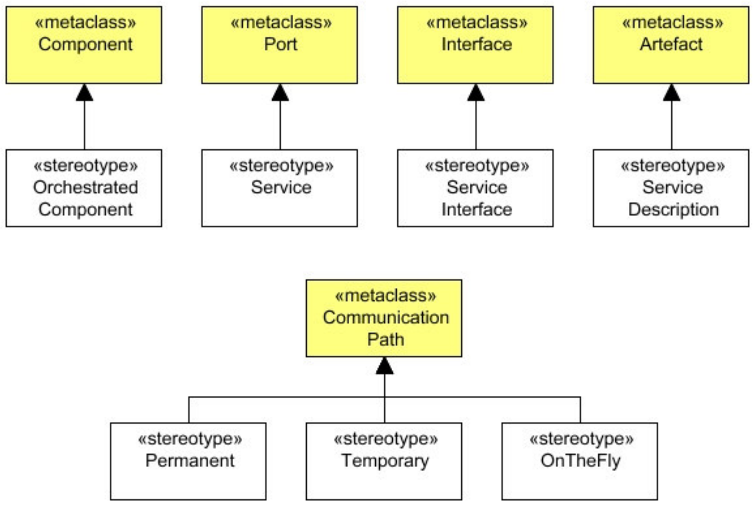

Figure 9: UML profile for service-oriented systems: structural elements

.

Stereotype name	UML base class	Description	Used in	Constraints
Orchestrated Component	Component	Component which is realised through orchestration.	Structure diagram	All ports are stereotyped services
Service	Port	Groups service provisions and requirements on services. Services are published, discovered, and bound	Structure diagram	
ServiceInterface	Interface	Interface specifying provided or required operations of a service.	Structure diagram	
ServiceDescription	Artefact	Characterization of the capabilities of a service that is defined according to standard languages and published according to standard protocols.	Structure diagram	
Permanent	CommunicationPath	Permanent communication paths between deployment targets for the exchange of messages and signals.	Deployment diagram	
Temporary	CommunicationPath	Temporary communication paths between deployment targets for the exchange of messages and signals.	Deployment diagram	
OnTheFly	CommunicationPath	Communication paths between deployment targets that require a discovery process for the exchange of messages and signals.	Deployment diagram	

Table 1: UML4SOA concepts of package Structure

3.2 UML Extension for Behavioural Modelling of Services

The UML profile for service-oriented systems includes the constructs for modelling the behaviour of SOAs, i.e. stereotypes for service-specific use cases, orchestration, compensation, and exchange of messages (service interaction).

Figure 10 depicts the UML structure diagrams showing these elements of the UML4SOA profile and the corresponding UML elements, for which they define an extension. Table 2 provide a description of each stereotype.

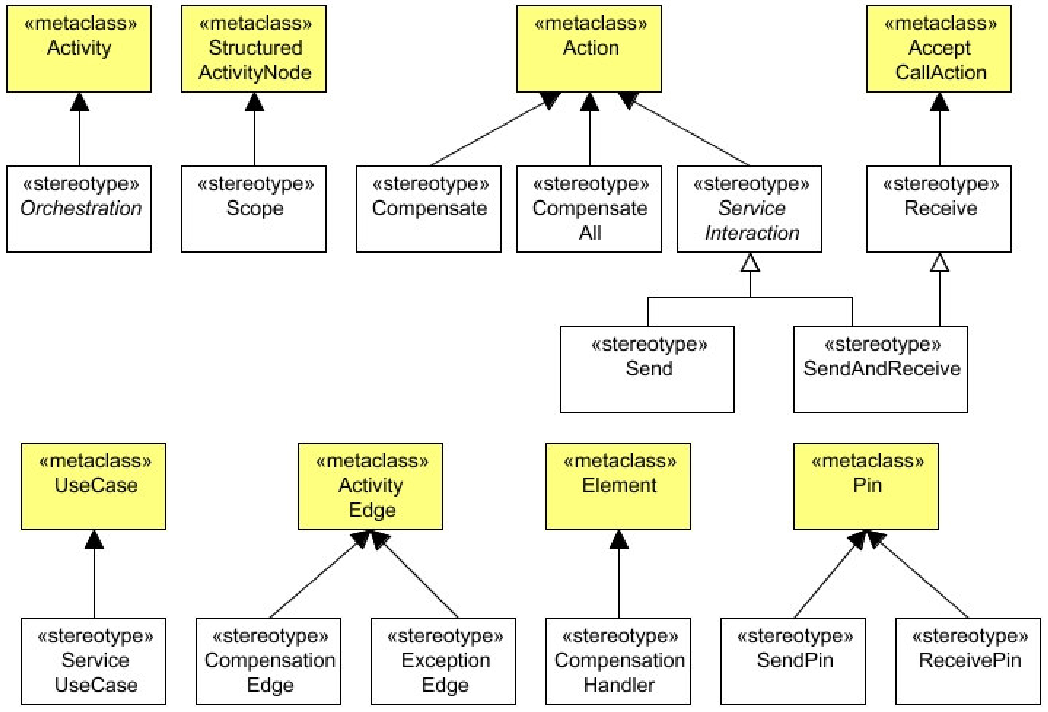
«metaclass» Activity
«metaclass» Structured ActivityNode
«metaclass» Action
«metaclass» Accept CallAction
«stereotype» Orchestration
«stereotype» Scope
«stereotype» Compensate
«stereotype» Compensate All
«stereotype» Service Interaction
«stereotype» Receive
«stereotype» Send
«stereotype» SendAndReceive
«metaclass» UseCase
«metaclass» Activity Edge
«metaclass» Element
«metaclass» Pin
«stereotype» Service UseCase
«stereotype» Compensation Edge
«stereotype» Exception Edge
«stereotype» Compensation Handler
«stereotype» SendPin
«stereotype» ReceivePin

Figure 10: UML profile for service-oriented systems: behavioural elements

Stereotype name	UML base class	Description	Used in	Constrains
Orchestration	Activity	Activity that specifies the orchestration of services to be executed.	– (Abstract class)	
Scope	Structured ActivityNode	Activity group that may be have associated event, exception and compensation handlers.	Activity diagram	
ServiceInteraction	Action	Interaction for the communication between partners.	– (Abstract class)	
Send	CallAction	Action that invokes behaviour sending a message synchronously, i.e. waits until the corresponding partner receives the message.	Activity diagram	
Receive	AcceptCallAction	Reception of a message. It blocks until a message is received.	Activity diagram	
SendAndReceive/ SendReceive	CallAction	Is a shorthand for a sequential order of send and receive actions.	Activity diagram	
SendPin	Pin	A pin that holds output values produced by an action (UML).	Activity diagram	

Stereotype name	UML base class	Description	Used in	Constrains
ReceivePin	Pin	A pin that holds input values to be consumed by an action (UML).	Activity diagram	
ExceptionEdge	ActivityEdge	Edge used to associate exception handlers to activities and scopes.	Activity diagram	
CompensationEdge	ActivityEdge	Edge used to associate compensation handlers to activities and scopes.	Activity diagram	
Compensate	Action	Action that triggers the execution of the compensation defined for a scope or activity.	Activity diagram	
CompensateAll	Action	Action that triggers compensation of the actually compensated scope (i.e. calling compensation on all subscopes in the order of their completion). Can be inserted in compensation handlers only.		
CompensationHandler	Element	Handler that specifies a body to execute in case a compensation is triggered by a compensate action.	Activity diagram	
ServiceUseCase	Use Case	Use case (functionality of the system) that will be offered as service, i.e. it will be published, discovered and other parties will be bound to the service when implemented.	Use Case Diagram	

Table 2: UML4SOA concepts of package Behaviour

3.3 UML Extension for Business Policies

Most of the concepts in the metamodel outlined in Section 2.3 can be rendered in standard UML provided that a suitable *framework* is defined, as we do in the end of this section. The main point to be addressed remains the representation of the tasks, i.e. of their behaviour in relation to policies. UML provides two ways, namely two Actions, to model the invocation of a behaviour in an Activity model: *CallOperationAction* and *CallBehaviorAction*. The latter is more direct, but *CallOperation* is more apt to our purposes. Indeed, this action calls an operation of an object. It is straightforward to associate the intended behaviour, shown in Figure 11, to such an operation. Moreover, the object can be used to hold the task attributes. Finally, the action input/output pins can hold the arguments/results of the operation.

The graphical representation of a task then is the standard one (see Figure 18), where the stereotype means that the invoked behaviour is the one defined above: The second line node in the workflow (to distinguish two occurrences of the same task type in a workflow), and the last line lists the type of the task (a related class diagram showing attributes may help here), and the called operation (taskBehaviour).

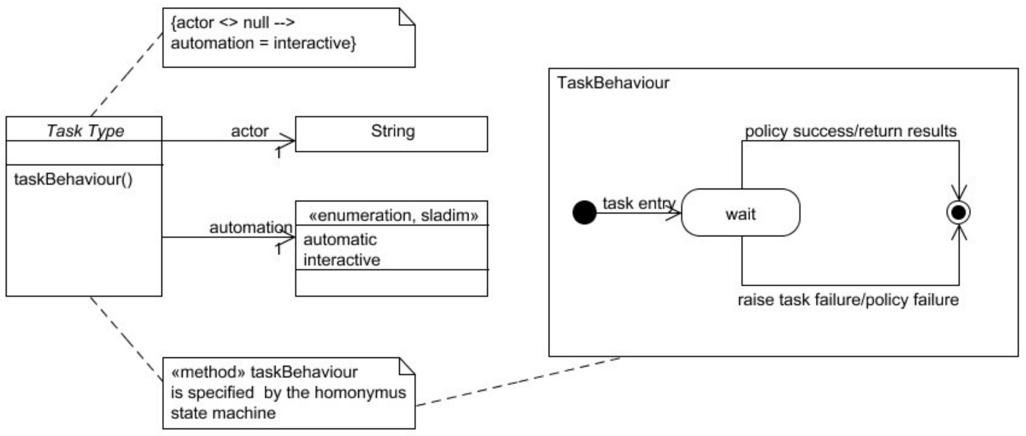

Figure 11: The StPowla framework

To complete the profile and obtain a modelling environment, we need also a framework. The StPowla profile deals with tasks, as elements of the workflows. It is specified in Table 3.

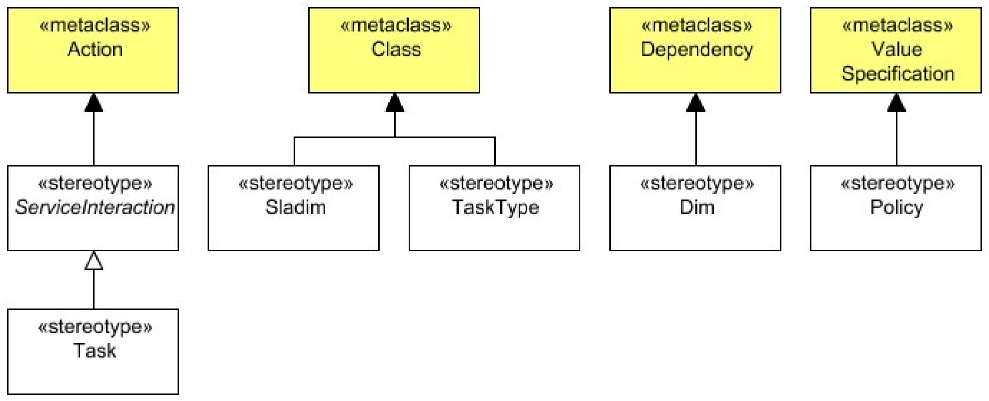

Figure 12: UML profile for service-oriented systems: business goals and policies

Stereotype name	UML base class	Description	Used in	Constrains
Task	Action	A task is a node in a workflow, executed by a service implementing the task type, activated via the policy associated to the node.	Activity diagram	The operation behaviour is taskBehaviour
TaskType	Class	A task type describe the type of a task	Class diagram	
Policy	ValueSpecification	A policy specifies choice of services	Class diagram	
Sladim	Class	These classes are used to specify the domain of SLAs.	Class diagram	
Dim	Dependency	The target specifies one of the SLA dimensions of the source.	Class diagram	The source is stereotyped TaskType, the target is stereotyped sladim

Table 3: UML4SOA concepts for business goals and policies

To understand the constraint in the first row, we need to consider, besides the profile, a (simple) *framework* that allows us to declare tasks, and so to deal with operations and attributes. Dealing with the attributes of the tasks requires more than the Actions, since in UML it is not possible to define attributes for Actions[1]. What we do then, is to define the task types as classes that inherit from a prototypical one, defined in the StPowlaFramework, as shown in Figure 11. Besides introducing two *general attributes* of the StPowla task types, i.e. automation and actor, the framework accounts for their standard behaviour. The task types used in the workflows will inherit this behaviour, but they can redefine its signature, to take care of the arguments and results of the specific requested service.

3.4 UML Extension for Non-Functional Properties of Services

The UML profile for service-oriented systems is enriched with constructs for modelling non-functional properties of services: stereotypes for the requester and provider specification and for the characteristics and dimensions of these characteristics. Monitoring of the contracts is also supported.

Figure 13 depicts the UML structure diagrams showing the elements the UML4SOA profile includes for modelling non-functional properties of services and the corresponding UML metaclasses, for which they define an extension. Table 2 provide a description of each stereotype.

[1] Or, at least the CASE Tool Rational Software Modeler does not support them.

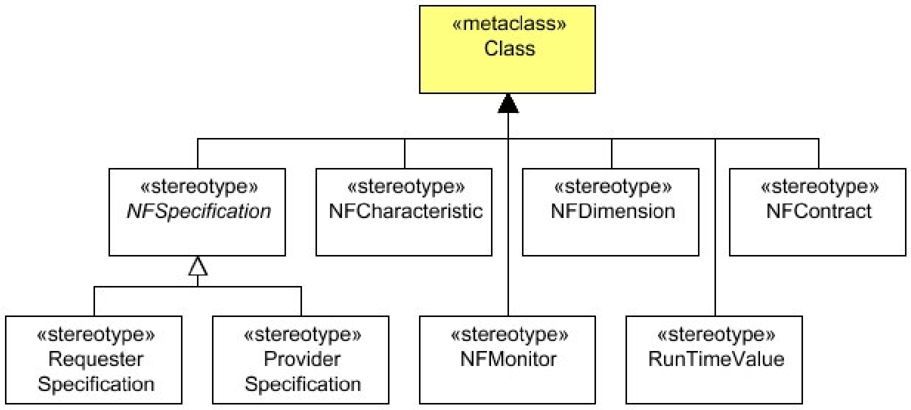

Figure 13: UML profile for service-oriented systems: Non-functional properties

Stereotype name	UML base class	Description	Used in	Constrains
NFCharacteristic	Class	Describes a certain non-functional aspect of a service, e.g., Performance.	Structure diagram	
NFSpecification	Class	Specifies non-functional requirements.	- (abstract class)	
NFContract	Class	A concrete contract between two components. Contains specifications, may be extended with additional information (e.g., obligations).	Structure diagram	There must be two services connected by "contractor" assoc, one having a RequesterSpecification and the other having a "ProviderSpecification".
NFDimension	Class	Determines a certain attribute (e.g., ResponseTime). May have additional attributes (such as deviation.etc.).	Structure diagram	
RunTimeValue	Class	Measured data; relation between RuntimeValue and NFDimension must be included in the contract.	Structure diagram	
NFMonitor	Class	External service to perform monitoring actions.	Structure diagram	
RequesterSpecification	Class	Specification describing requested functionality w.r.t non-functional characteristics.	Structure diagram	
ProviderSpecification	Class	Specification describing provided functionality w.r.t non-functional characteristics.	Structure diagram	

Table 4: UML4SOA concepts for non-functional properties of services

4 Modelling with UML4SOA

In this section the *On Road Assistance* scenario of the automotive case study is used to illustrate modelling with the UML4SOA profile. The scenario is described in deliverable D8.0 [GtBB+06]. For an overview of the specification of automotive scenarios, the reader is referred to deliverable D8.2a [KB2007] and for a detailed UML specification of the *On Road Assistance* scenario to [Ko2007].

In the *On Road Assistance* scenario, the diagnostic system reports a severe failure in the car engine; for example, the vehicle's oil lamp reports a low oil level. This triggers the in-vehicle diagnostic system to perform an analysis of the sensor values. The diagnostic system reports for example a problem with the pressure in one cylinder head, and therefore the car is no longer driveable, and sends a message with the diagnostic data and the vehicle's GPS data to the car manufacturer or service centre. Based on availability and the driver's preferences, the service discovery system identifies and selects the appropriate services in the area: garage (repair shop), tow truck and rental car. When the driver makes an appointment with the garage; the diagnostic data is automatically transferred to the garage, which could then be able to identify the spare parts needed to perform the repair. A towing service is also identified by the discovery system, providing the GPS data of the stranded vehicle. The driver makes an appointment with the towing service, and the vehicle is towed to the garage. The selection of services takes into account personalised policies and preferences of the driver. We assume that the owner of the car has to deposit a security payment before being able to order services.

4.1 Requirements Specification

Functional requirements are represented as uses cases. Figure 14 shows the **UML 2.0 Use Case Diagram**. Services are modelled using the stereotype «service use case»[2].

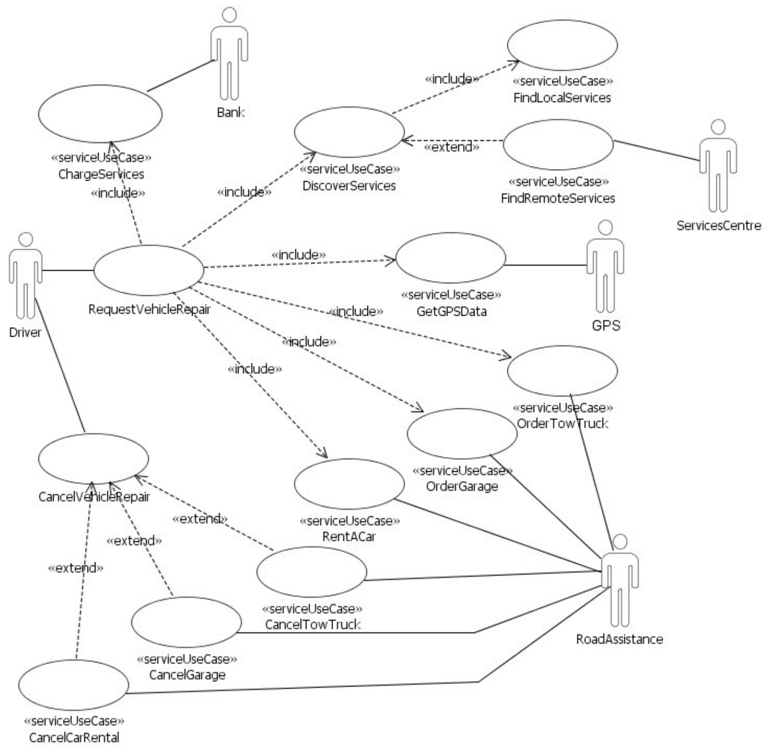

Figure 14: Use case diagram for the *On Road Assistance* scenario of the automotive case study

4.2 Architecture

Figure 15 depicts an overview of the architecture, which is represented as a UML 2.0 Deployment Diagram. The model shows the distribution of the components within the different physical devices of the vehicle. The UML4SOA stereotypes «temporary» and «on the fly» visualize temporary and on-the-fly communication paths. Communication of components within the vehicle usually uses permanent communication paths. The «permanent» stereotype is not visualized in order to avoid an overloaded diagram.

4.3 Structure and Behaviour

The *Orchestrator* is an architectural element that is in charge to achieve a goal by means of composition of services. Figure 16 shows the structure of the Orchestrator component and all components that the orchestrator needs to communicate with in order to orchestrate services in the *On Road Assistance* scenario. These components are the Reasoner and the Local Services, which are needed to discover services

[2] Stereotype names are written slightly different in the diagrams modelled by CASE tools that require Java-like notation for names.

(*findLocalServices*) and select services (*bestServices*) respectively. On the other hand, dynamic binding will be performed with external service providers such as a *Garage*, a *TowTruck* and a *RentACar,* which have been selected in the previous step according to the results of the discovery and reasoning process. For a more simple graphical representation, these services are accessible through the *Vehicle Communication Gateway*

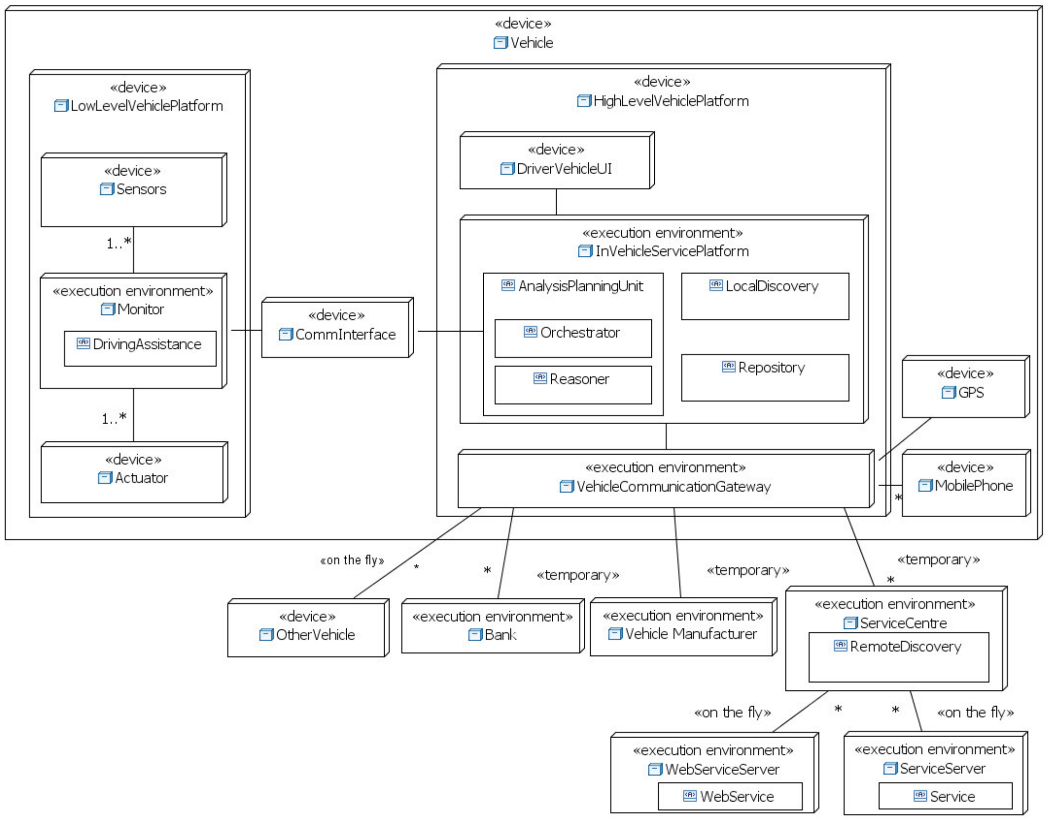
«device» Vehicle
«device» LowLevelVehiclePlatform
«device» Sensors
1..*
«execution environment» Monitor
DrivingAssistance
1..*
«device» Actuator
«device» CommInterface
«device» HighLevelVehiclePlatform
«device» DriverVehicleUI
«execution environment» InVehicleServicePlatform
AnalysisPlanningUnit
Orchestrator
Reasoner
LocalDiscovery
Repository
«device» GPS
«execution environment» VehicleCommunicationGateway
«device» MobilePhone
*
«on the fly»
*
*
«temporary»
«temporary»
«temporary»
*
«device» OtherVehicle
«execution environment» Bank
«execution environment» Vehicle Manufacturer
«execution environment» ServiceCentre
RemoteDiscovery
«on the fly»
*
*
«on the fly»
«execution environment» WebServiceServer
WebService
«execution environment» ServiceServer
Service

Figure 15: UML deployment diagram for the Architecture of the automotive case study

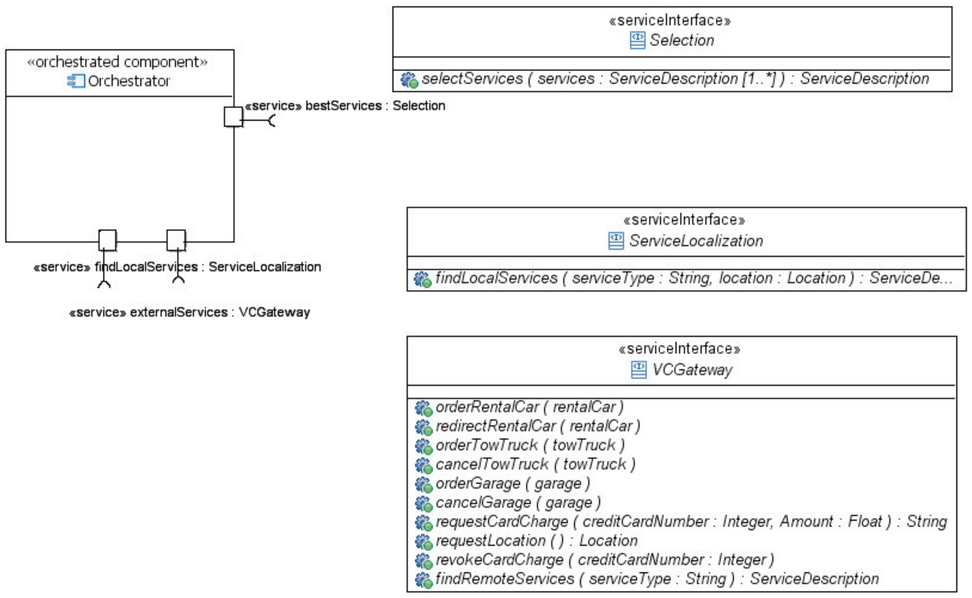
«orchestrated component» Orchestrator
«service» bestServices : Selection
«service» findLocalServices : ServiceLocalization
«service» externalServices : VCGateway
«serviceInterface» Selection
selectServices (services : ServiceDescription [1..*]) : ServiceDescription
«serviceInterface» ServiceLocalization
findLocalServices (serviceType : String, location : Location) : ServiceDe...
«serviceInterface» VCGateway
orderRentalCar (rentalCar)
redirectRentalCar (rentalCar)
orderTowTruck (towTruck)
cancelTowTruck (towTruck)
orderGarage (garage)
cancelGarage (garage)
requestCardCharge (creditCardNumber : Integer, Amount : Float) : String
requestLocation ( ) : Location
revokeCardCharge (creditCardNumber : Integer)
findRemoteServices (serviceType : String) : ServiceDescription

Figure 16: UML structure diagram for the *Orchestrator* component in the *On Road Assistance* scenario

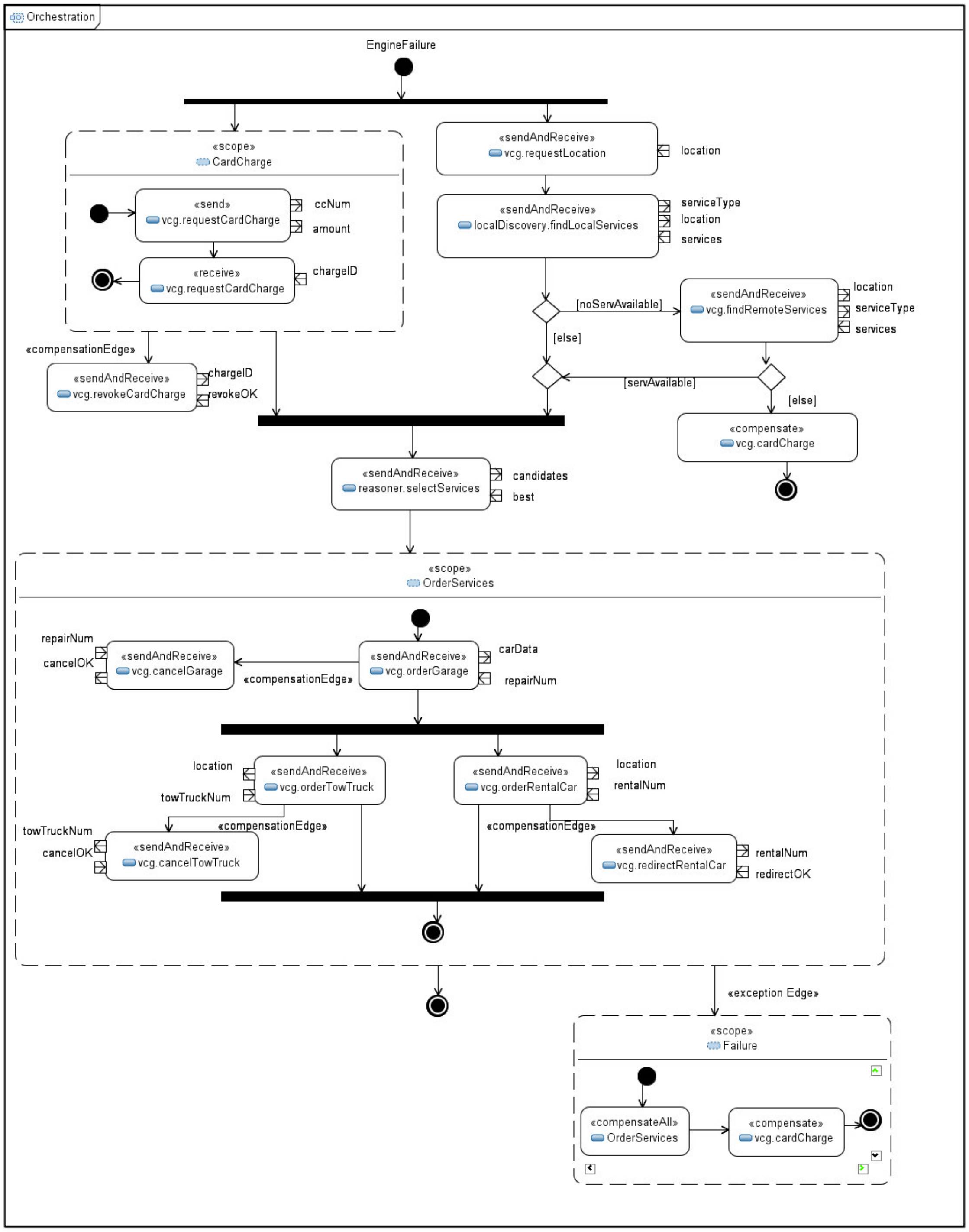
Orchestration
EngineFailure
«scope» CardCharge
«send» vcg.requestCardCharge
ccNum
amount
«receive» vcg.requestCardCharge
chargeID
«compensationEdge»
«sendAndReceive» vcg.revokeCardCharge
chargeID
revokeOK
«sendAndReceive» vcg.requestLocation
location
«sendAndReceive» localDiscovery.findLocalServices
serviceType
location
services
[noServAvailable]
«sendAndReceive» vcg.findRemoteServices
location
serviceType
services
[else]
[servAvailable]
[else]
«compensate» vcg.cardCharge
«sendAndReceive» reasoner.selectServices
candidates
best
«scope» OrderServices
repairNum
cancelOK
«sendAndReceive» vcg.cancelGarage
«compensationEdge»
«sendAndReceive» vcg.orderGarage
carData
repairNum
location
towTruckNum
«sendAndReceive» vcg.orderTowTruck
«sendAndReceive» vcg.orderRentalCar
location
rentalNum
«compensationEdge»
towTruckNum
cancelOK
«sendAndReceive» vcg.cancelTowTruck
«compensationEdge»
«sendAndReceive» vcg.redirectRentalCar
rentalNum
redirectOK
«exception Edge»
«scope» Failure
«compensateAll» OrderServices
«compensate» vcg.cardCharge

Figure 17: UML activity diagram for Orchestration of services in the *On Road Assistance* scenario

The process starts with a request from the Orchestrator to the Bank to charge the driver's credit card with the security deposit payment, which is modelled by an asynchronous UML action RequestCardCharge. To charge the credit card the card number is provided as a send parameter of the UML stereotyped call action. In general, to determine receive and send parameters, the following assignment is used: Receive pins have an arrow pointing towards to the action; send pins have an arrow pointing away from the action.

4.4 Business Goals and Policies

When looking at the car repair scenario from the business modelling perspective that is proper of StPowla, one has to concentrate on those facets that are relevant for the business workflow, as it appears to the final users. This attitude is different from the one of detailed service development, taken e.g. in the previous section. For instance, we assume that compensations will occur as needed, while compensations are a primary design concern there. In StPowla the scenario is described by a workflow, to which some policies apply.

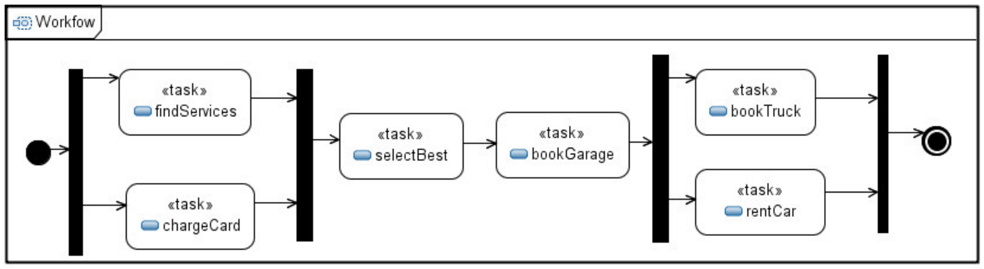

Figure 18: Car repair workflow

As an example of adaptation, we discuss a policy taking care of the fact that usually a driver already knows and trusts some of the garage and towing truck services of his own town.

P1: If the car fault happens in the driver's own town, then he wants to select the services to be used. Otherwise he wants them chosen automatically.

This policy can be expressed in StPowla as a policy which applies to task `selectBest`:

```
P1: appliesTo selectBest
       when taskEntry([])
         if findService.location = myTown
         do req(main, [], [Automation = interactive])
     seq
       when taskEntry([])
         do req(main, [], [Automation = automatic])
```

It is defined as a sequence, the first argument being the request of an interactive choice and the second one dealing with the general case. Operator `seq` checks its second argument only if the first one is not applicable, i.e. if the location is different from the driver's town. In the third and eighth lines, the StPowla action `req` looks for, binds and invokes a service of type `selectBest` with a SLA along the `Automation` dimension that ensures proper automation in the choice.

To make this policy expressible, the domain has to be specified appropriately, as shown by the class diagram in Figure 19: task type `SelectBest` has a SLA dimension, `Automation`. Any service which implements `selectBest`, that is, any service eligible for execution when the workflow reaches the task, must exhibit the ability to fulfil one of the foreseen automation modes. Then, the service request in the policy associated with the node will select one of the appropriate services. In addition, the task node `FindService` has an attribute, `location`, which carries the town where the car failure occurred (among other information, not specified here for sake of space).

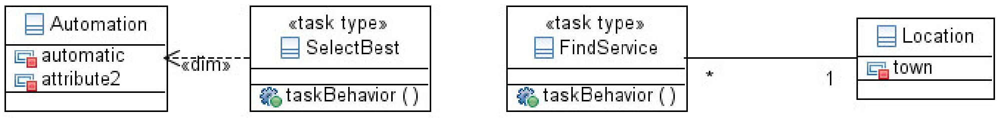

Figure 19: Domain for car repair (partial)

4.5 Non-functional Properties

The model presented here is part of the specification of *OnRoadAssistance* scenario of the automotive case study of SENSORIA. Parts of the structural model of this scenario were taken and extended with specifications, properties and contracts. Figure 20 shows *Vehicle Communication Gateway* component, and its specifications. This component as also tagged visually- provides some services (as a gateway for external communication) and requires some from external components, physically not contained within the car. As such, it has provider specifications and requester specifications as well. For the sake of visibility, specifications of two requested services (denoted by ports), namely those of *BankCharge* and *GPS* are included. To keep the diagram understandable, not all possible aspects are covered here, the aim was to show how different aspects (e.g., performance, secure communication, availability, etc.) can be incorporated using SENSORIA profile. Some example attributes are also listed to illustrate the detailed definition of *NFDimension*s, however, the list is not intended to be exhaustive. Usually, an SLA-like description includes not only one guaranteed value of a parameter, but minimum/maximum intervals, deviation, average, the length of the measurement interval, etc. This additional information can be added to concrete *NF Dimension*s, as shown by the example of *Throughput* and *ResponseTime*. Such models can be the basis of performance analysis and runtime monitoring as well.

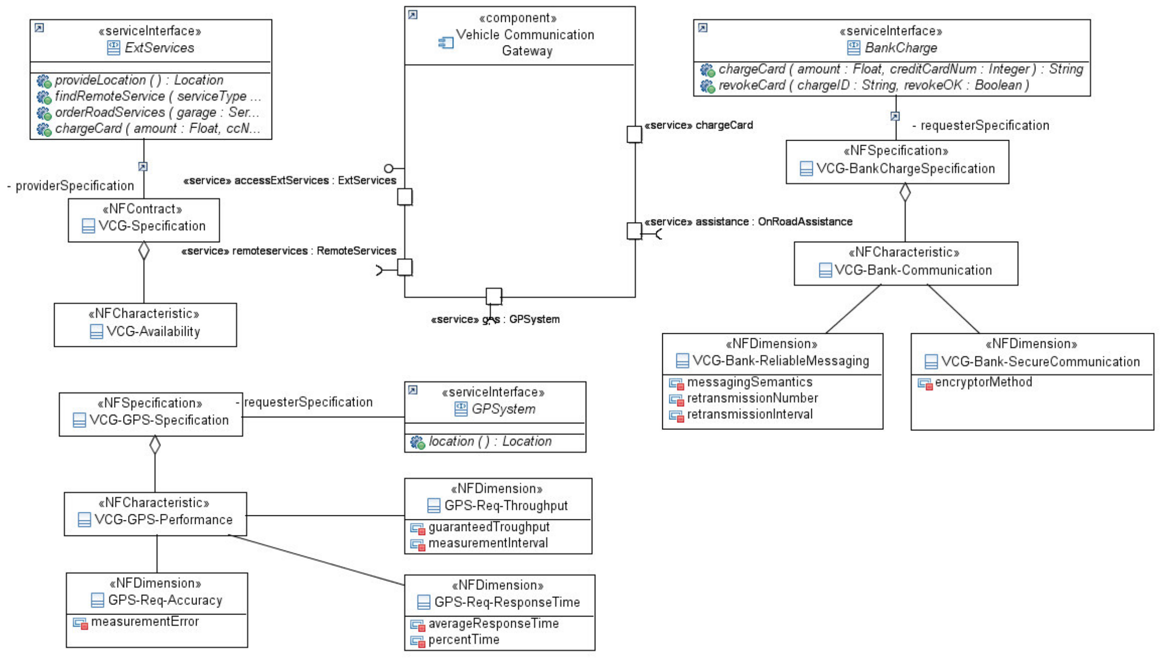
«serviceInterface» ExtServices
provideLocation ( ) : Location
findRemoteService (serviceType ...
orderRoadServices (garage : Ser...
chargeCard (amount : Float, ccN...
- providerSpecification
«NFContract» VCG-Specification
«NFCharacteristic» VCG-Availability
«component» Vehicle Communication Gateway
«service» accessExtServices : ExtServices
«service» remoteservices : RemoteServices
«service» gps : GPSystem
«service» chargeCard
«service» assistance : OnRoadAssistance
«serviceInterface» BankCharge
chargeCard (amount : Float, creditCardNum : Integer) : String
revokeCard (chargeID : String, revokeOK : Boolean)
- requesterSpecification
«NFSpecification» VCG-BankChargeSpecification
«NFCharacteristic» VCG-Bank-Communication
«NFDimension» VCG-Bank-ReliableMessaging
messagingSemantics
retransmissionNumber
retransmissionInterval
«NFDimension» VCG-Bank-SecureCommunication
encryptorMethod
«NFSpecification» VCG-GPS-Specification
- requesterSpecification
«serviceInterface» GPSystem
location ( ) : Location
«NFCharacteristic» VCG-GPS-Performance
«NFDimension» GPS-Req-Throughput
guaranteedTroughput
measurementInterval
«NFDimension» GPS-Req-Accuracy
measurementError
«NFDimension» GPS-Req-ResponseTime
averageResponseTime
percentTime

Figure 20: Specifications of non-functional properties for the *Vehicle Communication Gateway*

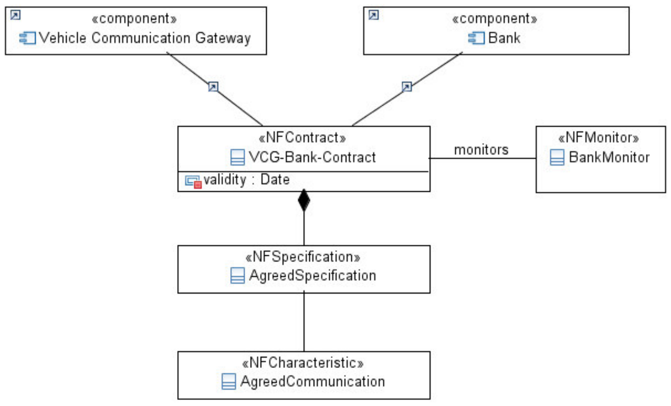
«component» Vehicle Communication Gateway
«component» Bank
«NFContract» VCG-Bank-Contract
validity : Date
monitors
«NFMonitor» BankMonitor
«NFSpecification» AgreedSpecification
«NFCharacteristic» AgreedCommunication

Figure 21: Contracts between the *Vehicle Communication Gateway* and the *Bank* components

5 Final Remarks

The SENSORIA UML profile defines services to be described by port protocols regulating the message exchange between service provider and requester. These protocols, typically specified as UML state machines or described by UML interactions, can be verified by using the tools developed in T3.5 for model checking UML designs. The current profile includes security annotations as non-functional properties, further security-related aspects will be pursued in T3.5.

There are some other approaches defining UML extensions for some aspects of service-oriented systems, such as [SCA07], [HLT03] and [UPMS]. UML4SOA is based on these approaches and the objective is to provide an integrate UML profile covering modelling requirements of structural and behavioural, policies and non-functional properties of SOAs.

To our knowledge, there is no much work on the use of UML in relation with policies. Class diagrams are used to relate the entities involved in Rule Based Access Control by Koch and Parisi-Presicce in [CPP03], where they use Alloy and graph-transformations for verification. Like we do, UML is used to provide the context for the use of policies, rather than to express them. Work in progress is the UML Profile and Metamodel for Services [UPMS] that is integrating a set of existing approaches that focus on different architectural styles. Based on the experience modelling with our UML4SOA we will be able to provide comments and suggestions for the UPMS approach, which is performed within the scope of an OMG standardisation process.

The experience using the UML4SOA profile in modelling the architectural and behavioural aspects of the *On Road Assistance* scenarios of the automotive case study and the experience done so far with the StPowla profile and framework seems promising. In particular, our results in modelling of service specific behaviour such as orchestration and compensation of services are interesting, but require further refinement. In addition, there is need of more work in two directions: more extensive modelling of case studies to straighten the details, and experimentation with UML design tools that fully support the profiling mechanism.

6 Relevant SENSORIA Publications and Reports

[BFGK06] Laura Bocchi, Alessandro Fantechi, László Gönczy, and Nora Koch. D1.1.a: Sensoria Ontology. Prototype Language for Service Modelling: Ontology for SOAs presented through Structured Natural Language. Technical report, 2006. Deliverable D1.1.a of SENSORIA.

[GMRFS07] Stephen Gorton, Carlo Montangero, Stephan Reiff-Marganiec and Laura Semini. Expressing essential requirements and variability of Business Processes. Third International Workshop on Engineering Service-Oriented Applications: Analysis, Design and Composition. Vienna, Sept. 17th, 2007. To appear.

[GRFT07] Stephen. Gorton, Stephan Reiff-Marganiec and Emilio Tuosto. Policy-driven Reconfiguration of Service-Targeted Workflows. SENSORIA Tech. Rep. July 2007.

[GtBB+06] Stefania Gnesi, Maurice ter Beek, Hubert Baumeister, Matthias Hoelzl, Corrado Moiso, Nora Koch, Angelika Zobel, and Michel Alessandrini. D8.0: Case Studies Scenario Description. Technical report, 2006. Deliverable D8.0 of SENSORIA.

[GV06] László Gönczy and Daniel Varró. Modeling of reliable messaging in service oriented architectures. In Andrea Polini, editor, Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006), pages 35–49, Palermo, Sicily, ITALY, June 9th 2006.

[HLT03] Reiko Heckel, Martin Lohmann and SebastianThöne Towards a UML Profile for Service-Oriented Architectures, Workshop on Model Driven Architecture: Foundations and Applications (MDAFA), CTIT Technical Report TR–CTIT–03–27, University of Twente, The Netherlands, June 2003

[Ko2007] Nora Koch: Automotive Case Study: UML Specification of On Road Assistance Scenario, SENSORIA Report, August 2007

[KB2007] Nora Koch and Dominik Berndl: Requirements Modelling and Analysis of Selected Scenarios Automotive Case Study Technical Report 2007. Deliverable D8.2a of SENSORIA

7 Other References

[CPP03]	Manuel Koch and Francesco Parisi-Presicce. Visual Specifications of Policies and their Verification. Proc. FASE 2003, 278-293, LNCS 2621, Springer.
[Obj05]	Object Management Group. UML Profile for Schedulability, Performance and Time, version 1.1. Technical Report, 2005. OMG Available Specification.
[Obj06]	Object Management Group. UML profile for modeling quality of service and fault tolerance characteristics and mechanisms, version 1.0. Technical report, 2006. OMG Available Specification.
[OH]	Guadalupe Ortiz and Juan Hernández. Toward uml profiles for web services and their extrafunctional properties. In IEEE International Conference on Web Services (ICWS'06).
[OMGP]	OMG: Catalog of UML Profiles. http://www.omg.org/technology/documents/profile_catalog.htm, last visit:1.10.2007
[SCA07]	SCA Consortium. SCA Policy Framework. Version 1.0. Technical report, 2007.
[UML]	OMG. Unified Modeling Language: Superstructure, version 2.1.1 formal/2007-02-05, 2007.
[UPMS]	OMG. UML Profile and Metamodel for Services, http://www.omg.org/docs/ad/07-06-02.pdf, last visit: 1.10.2007
[WSLA]	IBM. Web Service Level Agreements Project. http://www.research.ibm.com/wsla/, last visit:1.10.2007