Specification and Implementation of Demonstrators for the Case Studies[⋆]

Jannis Elgner[1], Stefania Gnesi[2], Nora Koch[3,4], and Philip Mayer[3]

[1] S & N AG, Germany
jelgner@s-und-n.de
[2] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
ISTI–CNR, Pisa, Italy
gnesi@isti.cnr.it
[3] Ludwig-Maximilians-Universität München, Germany
{kochn,mayer}@pst.ifi.lmu.de
[4] Cirquent GmbH, Germany

Abstract. A main challenge in Sensoria has been the inclusion of case studies from different industrial and academic application areas, namely finance, automotive, telecommunications, and university administration. The case studies, along with a short description of available scenarios, have already been introduced in Chapter 0-3. In this chapter, we go into more detail, presenting the (graphical) specifications for selected scenarios by using the modeling approaches introduced in Sensoria. Furthermore, we detail the implementation of demonstrators for some of the case studies.

1 Introduction

The partners of the Sensoria project have used realistic case studies for feeding and steering the research process according to the expectations of society and its economy, discussing and communicating ideas among partners and communicating research results to and getting feedback from the research community at large. These case studies have already been shortly introduced in Chapter 0-3. Each of the scenarios presented has been employed by different partners with different requirements, methods, and tools as a test bed for demonstrating the feasibility and effectiveness of the use of the Sensoria results.

In this chapter, we present some of the scenarios in more detail. We select three scenarios from the case studies which have been extensively used in the project with the application of research results and tools. For two of the scenarios, namely finance and eUniversity, we provide an extended description by using the graphical modeling languages used or introduced in Sensoria, namely the UML extensions UML4SOA [3] and the upcoming OMG standard SoaML [4]. For the automotive case study, we present a demonstrator, i.e., the software resulting from applying the Sensoria development approach to the development of a SOA system.

[⋆] This work has been partially sponsored by the project SENSORIA, IST-2005-016004.

The following three sections each present one of the scenarios; starting with Finance, moving on to Automotive, and finally discussing eUniversity. We conclude in Sect. 5.

2 Finance Case Study: Credit Request Scenario

The *CreditRequest* scenario [1] from the finance domain models the loan approval workflow of a bank: A customer intends to lend some money, i.e. request a credit. During the process of approving or disapproving the credit request process, the customer must provide some input (like balances and securities), and the bank must either automatically or via human intervention approve or decline the request. A *risk rating* determines most of the decisions during this process, for example, whether a credit request is approved at all, and whether it can be approved automatically.

This scenario has been modeled in Sensoria with a combination of *SoaML* and *UML4SOA* elements, and has been implemented using model transformations to BPEL and WSDL code. In this section, we introduce parts of the model for the *CreditRequest* scenario.

Fig. 1 shows the static system structure of the scenario. The main process, shown in the middle and implemented as an orchestration, is the *CreditRequest*, which provides its services through the *CreditManagementService* port. *Rating* is another orchestration which the *CreditRequest* participant uses to calculate the rating. The services of *Rating* itself are provided through the *RatingService* port (left of the *Rating* participant).

The other participants are atomic services performing tasks like calculating ratings, storing data, and interacting with the user.

– The *Portal* services, both provider and consumer, are services concerned with user interaction. They are implemented as a set of web pages which handle communication with the customer and the bank employees through different frontends.
– The *CustomerManagement* service provides an interface to the customer database for identifying customers.
– For analysing input data from the customer, the two services *SecurityAnalysis* and *BalanceAnalysis* are used.
– Finally, the *RatingCalculator* service is used to calculate the actual risk rating.

The entire workflow implemented in the *CreditRequest* and *Rating* orchestrations has been defined with the help of the UML4SOA UML extension, and used as input to verification tools and for generating code. Due to the size of the process it is not possible to show every part here; we therefore have to content ourself with the overview of the behavior of the *CreditRequest* process as shown in Fig. 2.

The process is further subdivided into individual service activities which we shall describe now.

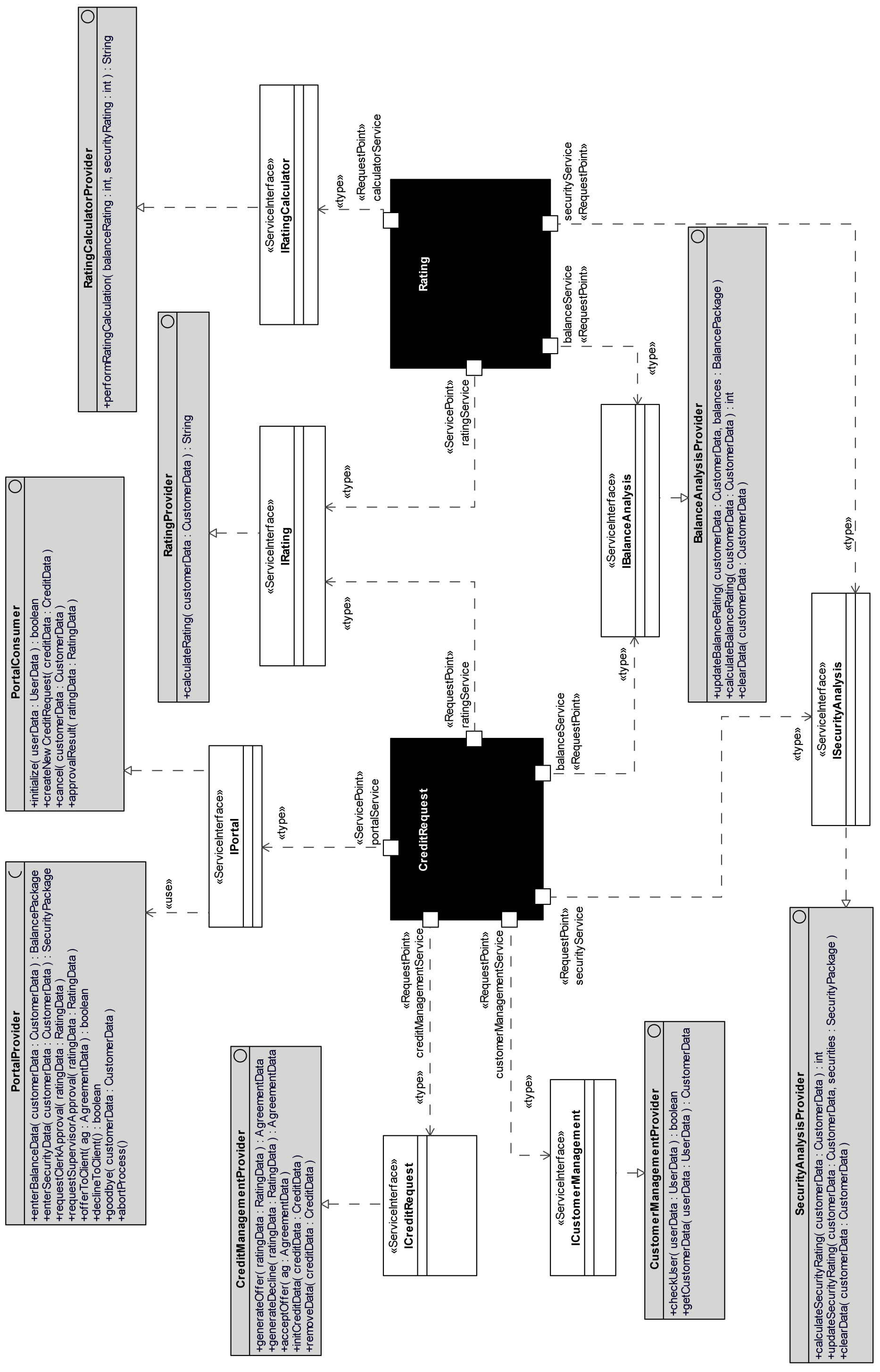

RatingCalculatorProvider
+performRatingCalculation(balanceRating : int, securityRating : int) : String
«ServiceInterface»
IRatingCalculator
«type»
«RequestPoint»
calculatorService
Rating
securityService
«RequestPoint»
balanceService
«RequestPoint»
«type»
RatingProvider
+calculateRating(customerData : CustomerData) : String
«ServicePoint»
ratingService
«ServiceInterface»
IRating
«type»
«type»
«ServiceInterface»
IBalanceAnalysis
BalanceAnalysisProvider
+updateBalanceRating(customerData : CustomerData, balances : BalancePackage)
+calculateBalanceRating(customerData : CustomerData) : int
+clearData(customerData : CustomerData)
«type»
PortalConsumer
+initialize(userData : UserData) : boolean
+createNewCreditRequest(creditData : CreditData)
+cancel(customerData : CustomerData)
+approvalResult(ratingData : RatingData)
«RequestPoint»
ratingService
balanceService
«RequestPoint»
«type»
«ServiceInterface»
ISecurityAnalysis
«type»
«ServiceInterface»
IPortal
«type»
«ServicePoint»
portalService
CreditRequest
«use»
PortalProvider
+enterBalanceData(customerData : CustomerData) : BalancePackage
+enterSecurityData(customerData : CustomerData) : SecurityPackage
+requestClerkApproval(ratingData : RatingData)
+requestSupervisorApproval(ratingData : RatingData)
+offerToClient(ag : AgreementData) : boolean
+declineToClient() : boolean
+goodbye(customerData : CustomerData)
+abortProcess()
«RequestPoint»
creditManagementService
«type»
«RequestPoint»
customerManagementService
«RequestPoint»
securityService
«type»
CreditManagementProvider
+generateOffer(ratingData : RatingData) : AgreementData
+generateDecline(ratingData : RatingData) : AgreementData
+acceptOffer(ag : AgreementData)
+initCreditData(creditData : CreditData)
+removeData(creditData : CreditData)
«ServiceInterface»
ICreditRequest
«ServiceInterface»
ICustomerManagement
CustomerManagementProvider
+checkUser(userData : UserData) : boolean
+getCustomerData(userData : UserData) : CustomerData
SecurityAnalysisProvider
+calculateSecurityRating(customerData : CustomerData) : int
+updateSecurityRating(customerData : CustomerData, securities : SecurityPackage)
+clearData(customerData : CustomerData)

Fig. 1. The Static Structure of the CreditRequestScenario.

- The *Initialize* service activity is used to bootstrap the service. A customer logs into the website, which leads to a call from the portal to the *CreditRequest* orchestration. The credentials of the customer are verified and, in the positive case, he is logged in to the system.
- We are then entering a loop in which a credit may be requested more than once by the same user with a changing amount of money or different securities and balances. The loop contains four activities: *Creation*, *HandleBalance&SecurityData*, *RatingCalculation*, and *Decision*. During the execution, an event may occur (*Cancel*), which aborts the process. Also, an exception might be thrown which is handled in *MainFault* and also leads to termination of the process.
- The *Creation* scope deals with initializing the workflow: A request for a new credit is received and the data initialized.
- *HandleBalance&SecurityData* handles the upload and storage of the balances and securities of the user. The balances are stored in the *BalanceService* and the securities are stored in the *SecurityService* for later retrieval by the *Rating* orchestration.
- In *RatingCalculation*, the second orchestration *Rating* is invoked to provide the main workflow with a risk rating, identifying the risk involved with the requested credit. This rating implies whether the request can be accepted automatically. If not, the rating implies whether the decision can be made by a clerk or has to be escalated to a supervisor.
- The *Decision* activity handles the process of deciding first whether the bank accepts the credit request, and second allows the customer to review, accept, or reject the offer. The first part, if not done automatically, involves a call to the portal which enables the corresponding bank employee to review the credit request and give his input. The second also involves the portal; this time the customer is notified and can review an offer on the website.
- If the customer has accepted an offer, or is no longer interested, the *Finalize* activity cleans up and ends the process.

The *CreditRequest* scenario has been used as input for the formal verification tools in Sensoria. This is described in detail in Chapter 7-4.

3 Automotive Case Study: On Road Assistance

In the *OnRoadAssistance* scenario from the automotive case study [2,5] the vehicle reacts to a failure in the car engine. Such an event triggers the in-vehicle diagnostic system to perform an analysis of the sensor values. The diagnostic system reports e.g. a problem with the pressure in one cylinder head, indicating that the driver will not be able to reach the planned destination. The necessary reactions to this report are handled in a service-oriented way by means of an orchestration.

Like the finance scenario from the last section, the *OnRoadAssistance* scenario has been modeled with a combination of the *SoaML* and *UML4SOA* extensions to the UML. Fig. 3 shows the orchestration of services for the *OnRoadAssistance* participant. See Chapter 1-1 for further details of both the static

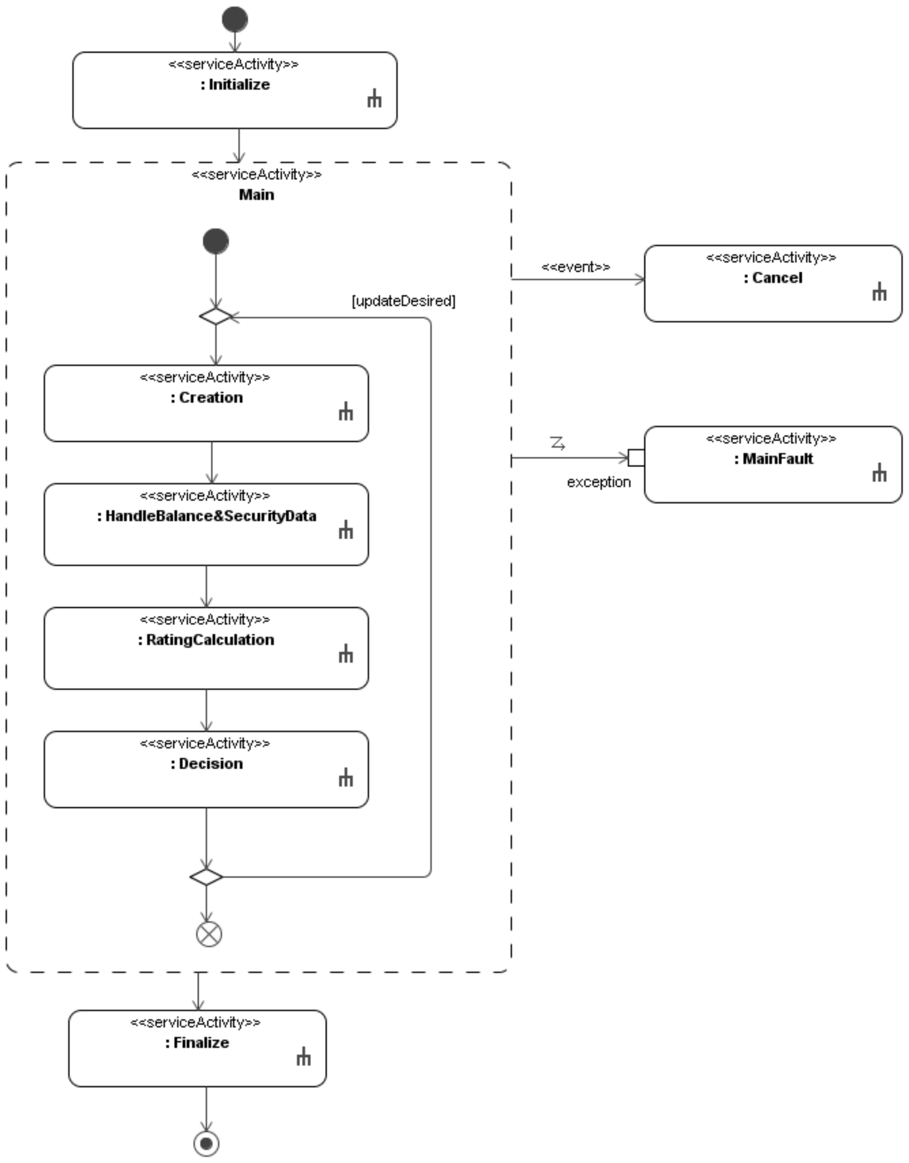


Fig. 2. The behavioral specification of the CreditRequest scenario.

structure and the dynamic behavior of this scenario, which we will not repeat here. Instead, we discuss the *implementation* of a *demonstrator* for the scenario on the technical basis of web services, which is achieved by means of automated model transformations based on other results of the Sensoria project.

The main goal of the automotive demonstrator [5] is to show the power of the Sensoria development approach based on the application of model-driven architecture (MDA) principles in the area of service-oriented computing. Following the model-driven approach, an implementation is created by the construction of models and model transformations. The demonstrator shows how this model-driven development process which automatically generates and deploys a service based on a model of the composition of services works in practice.

The automotive demonstrator for the *OnRoadAssistance* scenario is built as a web application: On the client side, only a fully JavaScript enabled web browser is needed. The business logic, specified as a service orchestration, is deployed on the server side. On the server, the application uses the following three tier architecture (see also Fig. 4).

Presentation Layer. The ViewManager in the presentation layer is a component developed to parse the client request, to call the service orchestration and generate web pages for the client.

Business Logic Layer. The business logic is located in the second layer. The main component is a BPEL process which is in charge of the service orchestration. Several local or remote web services can be called by the BPEL process. A special service is used for the invocation of a broker for dynamically identifying partner services (we use the broker Dino, which has been developed within Sensoria).

Database Layer. A database lies in the third layer, i.e., the persistence layer. The database contains all data needed by the services.

The Automotive Demonstrator implements the *OnRoadAssistance* scenario [2]. In order to keep the scenario simple, the demonstrator is limited to localizing garage and rental car station services, but this can be easily extended e.g. to identify as well a towing service, providing the GPS data of the stranded vehicle in case the vehicle is no longer drivable.

The services involved in the implementation of the *OnRoadAssistance* are the following:

– A *Position Service* providing the GPS data of the stranded vehicle
– A *Bank Service* for charging a credit card
– *Garage Services* for localizing and selecting garages
– *Rental Car Services* for localizing and selecting car rental stations

For demonstration purposes, two models of the same process are built, showing the benefits of the model-driven approach by later switching between them.

The first model is built as the sequential orchestration of the required services for a) determining the car position, b) finding garages in the vicinity of the car

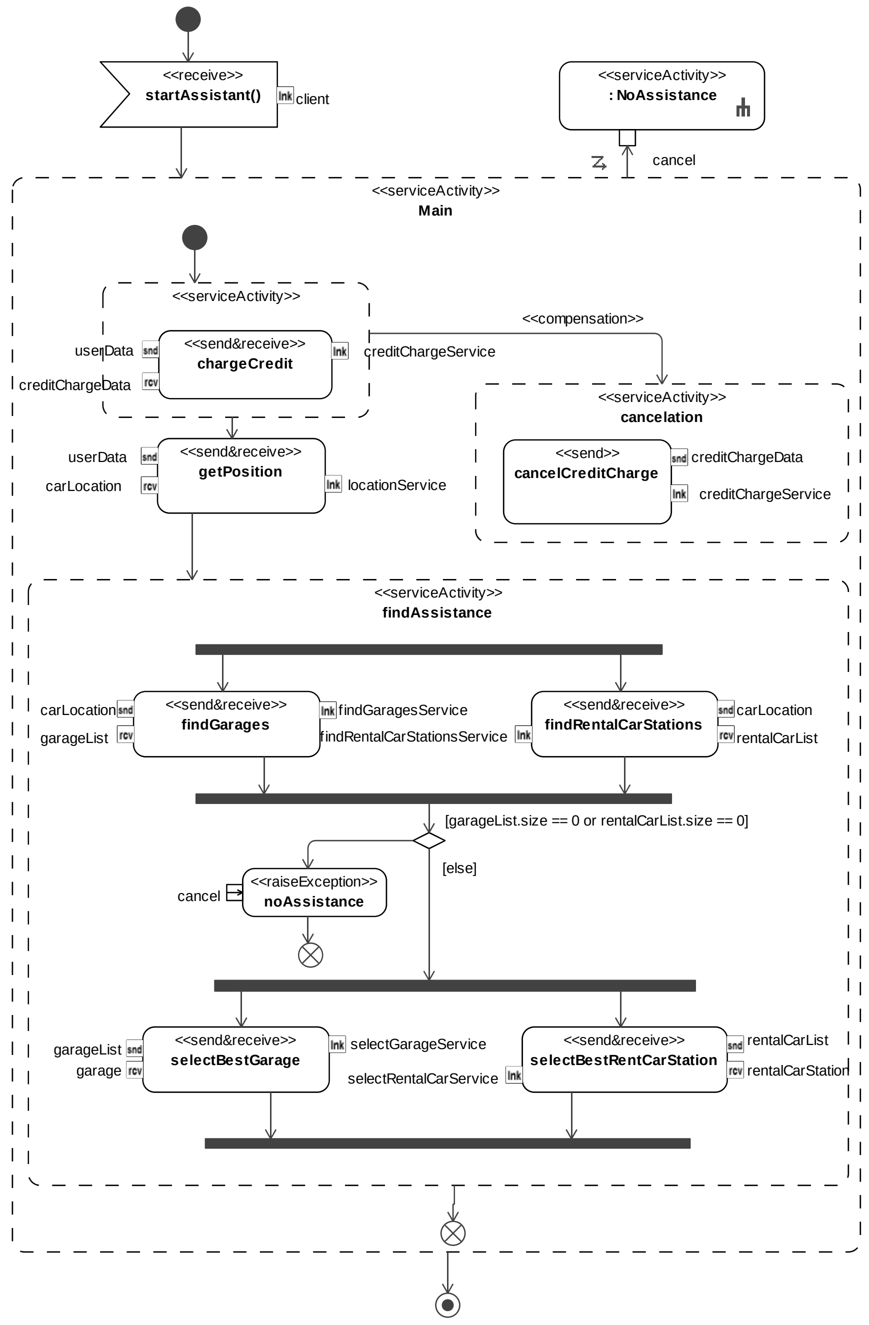


Fig. 3. UML4SOA activity diagram showing the OnRoadAssistance participant.

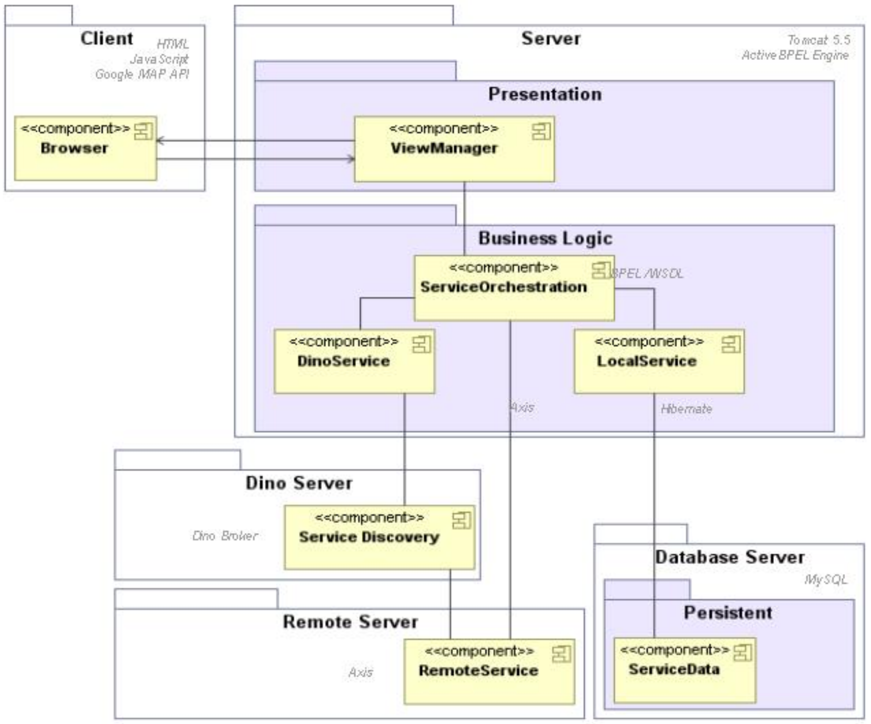
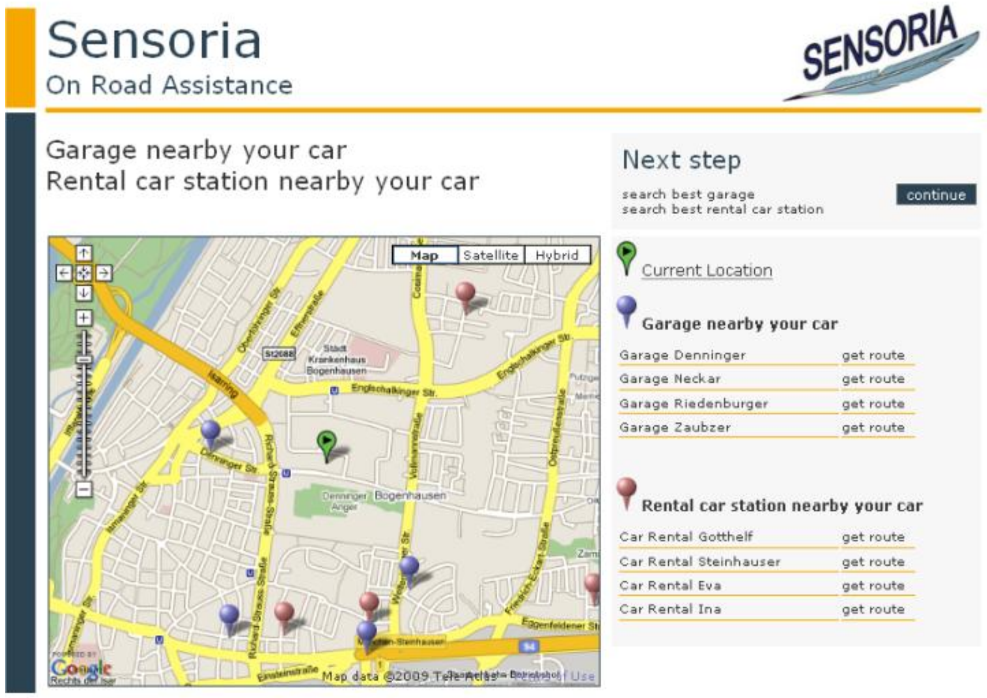


Fig. 4. Architecture of the automotive demonstrator.

and selecting the most convenient garage, and c) finding rental car stations nearby and selecting one. The orchestration process finishes with the credit card charge service. Using a chain of model transformations, the model is transformed to an executable service implemented in BPEL and deployed to an appropriate application server. Model transformations and deployment are performed in a fully automatic way.

The automotive demonstrator is designed in such a way that the invocation of each service is visualized in a web browser and expects a user interaction, almost all just a click on a continue button. In fact, the position of the car, asset of garages and car rentals nearby the car position, and then the selected garage and car rental station are visualized using the Google maps API (see Fig. 5). For the implementation of the interactions and the visualisation, dynamically generated web pages are associated to each service, and the BPEL process is enriched with additional interactive features by a complementary model transformation.

The power of the model-driven development approach is shown by a second run of the generation that consists of changing the orchestration model shown in Fig. 3. The changes are twofold: First, the credit charge service is invoked at the beginning of the process instead of at the end, and second, the localisation


Sensoria
On Road Assistance

Garage nearby your car
Rental car station nearby your car

Next step
search best garage
search best rental car station

Current Location

Garage nearby your car

Garage Denninger	get route
Garage Neckar	get route
Garage Riedenburger	get route
Garage Zaubzer	get route

Rental car station nearby your car

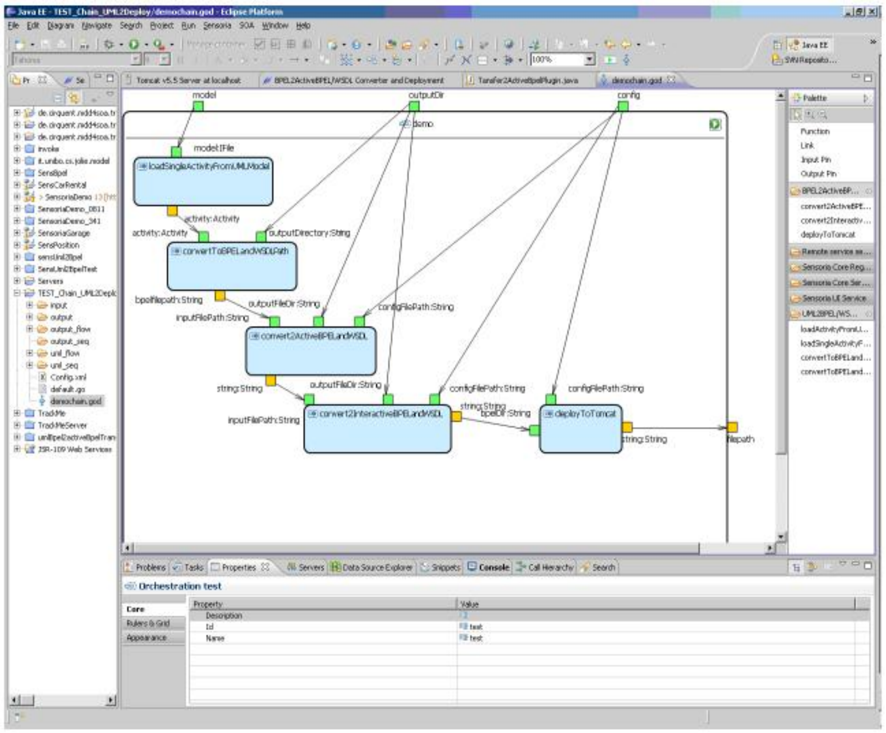
Car Rental Gotthelf	get route
Car Rental Steinhauser	get route
Car Rental Eva	get route
Car Rental Ina	get route

Fig. 5. Screenshot of the Automotive Demonstrator.

of garages and rental car stations as well as the selection of the most appropriate garage and rental car station are parallelized (see Fig. 5).

The process of generating the automotive demonstrator was implemented using the Sensoria Development Environment (SDE), which is an Eclipse-based framework for the integration and use of the tools developed in the project for the analysis and development of service-oriented software. The model transformations and deployment are performed automatically with the same tool chain that was implemented for this purpose in the SDE (see Fig. 6). For further details on SDE the reader is referred to Chapter 6-5.

The basic transformation from SoaML and UML4SOA to BPEL and WSDL is performed by using a first transformer from the MDD4SOA suite [3]. Additional model-to-model and model-to-code transformers of MDD4SOA have been introduced to handle user interactions and automatic deployment of a web application onto a web application server. The first additional model transformation is needed to provide BPEL and WSDL code that is executable by a specific BPEL engine (in our case, ActiveBPEL). The second model transformation is needed to allow user interactions and to visualize results step by step during the demonstration. The third one is used to deploy the resulting web application. Note that the model transformations are independent of the scenario, even more



Property	Value
Description	
Id	test
Name	test

Fig. 6. SDE tool chain for the model-driven development.

they are independent of the BPEL/WSDL application, i.e. they are generic and reusable for other services modeled as orchestration of other services.

4 eUniversity Case Study: Student Application

In the *StudentApplication* scenario of the eUniversity case study, students may apply for a certain course of studies at a university online, providing the necessary documents and certificates via a website. The functionality for handling applications is provided by service orchestrations which make use of a number of atomic services like the student office service, an admission checking service, and a service for the upload of documents to perform their task.

In the following, we detail the model of the *StudentApplication* scenario. Again, we employ the UML modeling language with the additional profiles SoaML [4] and UML4SOA [3] presented in Chapter 1-1. Like the other scenarios presented in the last sections, the student application scenario has been used as input for verification tools in Sensoria and has been implemented on the basis of Web Service technology.

The components of the eUniversity case study which are relevant for the student application scenario are shown in Fig. 7. The figure shows the overall

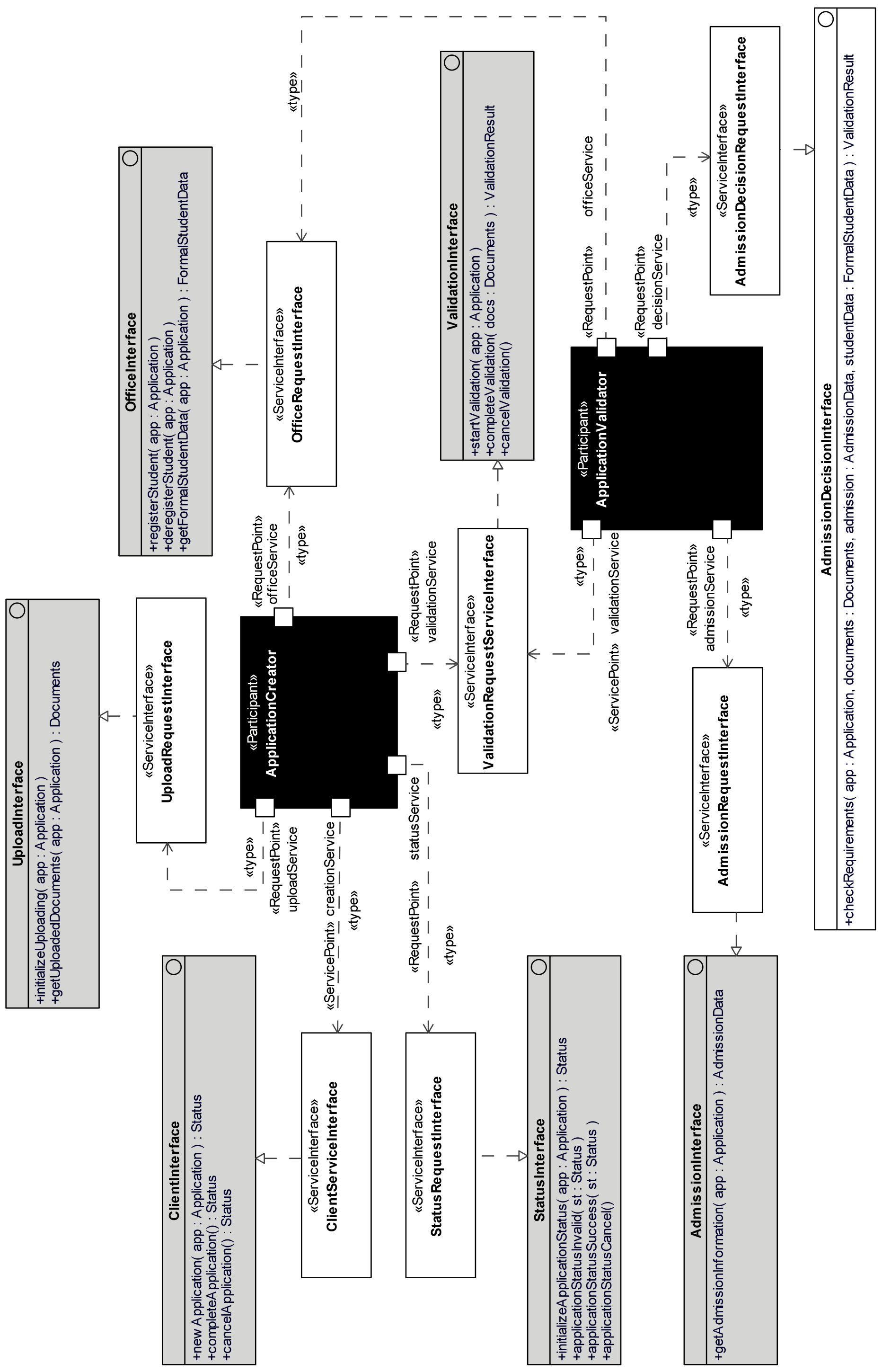

«Participant» ApplicationCreator
«Participant» ApplicationValidator
«ServiceInterface» UploadRequestInterface
«ServiceInterface» OfficeRequestInterface
«ServiceInterface» ValidationRequestServiceInterface
«ServiceInterface» AdmissionDecisionRequestInterface
«ServiceInterface» AdmissionRequestInterface
«ServiceInterface» ClientServiceInterface
«ServiceInterface» StatusRequestInterface
UploadInterface
+initializeUploading(app : Application)
+getUploadedDocuments(app : Application) : Documents
OfficeInterface
+registerStudent(app : Application)
+deregisterStudent(app : Application)
+getFormalStudentData(app : Application) : FormalStudentData
ValidationInterface
+startValidation(app : Application)
+completeValidation(docs : Documents) : ValidationResult
+cancelValidation()
AdmissionDecisionInterface
+checkRequirements(app : Application, documents : Documents, admission : AdmissionData, studentData : FormalStudentData) : ValidationResult
ClientInterface
+newApplication(app : Application) : Status
+completeApplication() : Status
+cancelApplication() : Status
StatusInterface
+initializeApplicationStatus(app : Application) : Status
+applicationStatusInvalid(st : Status)
+applicationStatusSuccess(st : Status)
+applicationStatusCancel()
AdmissionInterface
+getAdmissionInformation(app : Application) : AdmissionData
«RequestPoint» uploadService
«ServicePoint» creationService
«RequestPoint» statusService
«RequestPoint» officeService
«RequestPoint» validationService
«ServicePoint» validationService
«RequestPoint» admissionService
«RequestPoint» officeService
«RequestPoint» decisionService
«type»

Fig. 7. The eUniversity StudentApplication scenario.

composition of the SOA system modeled as a UML class diagram using SoaML model elements. Each of our two orchestrations offers or requires multiple services: The *ApplicationCreator* is invoked by the client for the creation of a new application, but invokes several other services as well, such as the *ValidationService* and the *StatusService*. The objective of the *ApplicationValidator* is to verify whether the application follows the policies of the university. The actual implementation of the two orchestrations further refines the behaviour of this scenario. The other services, including the client service, are atomic and implemented in a standard programming language (for example, in Java).

Overall, the scenario works as follows: A student uses the website to apply for a certain course of studies. The website (not shown) contacts the *ApplicationCreator* through its *creationService* service port. The *ApplicationCreator*, in turn, calls other entities through the *uploadService*, the *officeService*, and the *statusService* ports. Last but not least, it also contacts the *ApplicationValidator* through the *validationService* port for checking the student data and setting the status of the application. Being implemented as an orchestration itself, the *ApplicationValidator* works with other entities too – through the *officeService* (again), the *admissionService*, and finally the *decisionService* ports – to carry out the validation task. After a review of the application by the various services, the student is notified whether he was accepted at the university.

The two processes *ApplicationCreator* and *ApplicationValidator* from Fig. 7 are modeled as UML4SOA orchestrations. The first one is shown in Fig. 8. It illustrates how the creator interacts with its partners through ports. It starts with a receipt of the call *newApplication* through the *creationService* service port, receiving the application. After the receipt of this call, the *StatusService* and the *UploadService* are initialized, and the initial call is returned. Completing the initialisation phase, the *startValidation* call is sent to the *ApplicationValidator* to request the start of the validation. After having done so, the process waits for another call from the client. The student will either press the button to complete the application, or another one to cancel it.

If a *cancelApplication* call is received, the validation service is instructed to cancel the validation, and the status service is notified that the application has been canceled. If, on the other hand, the student chose to complete the application, the uploaded documents are retrieved from the *uploadService* and a final validation is requested from the *ApplicationValidator*, using the *completeValidation* call. If the result is okay, the student is registered at the *StudentOffice* with *registerStudent*. In any case, the initial call is replied to.

Besides the normal flow of the activity, the diagram also shows a second structured activity node – a compensation handler. The actions defined within *CompensationHandler* are executed if the main activity has been completed successfully, but needs to be undone. This functionality can be triggered externally after the orchestration has been completed. If the application has been completed successfully before, the student is removed from the list of applicants by using a *deregisterStudent* call on the *OfficeService*.

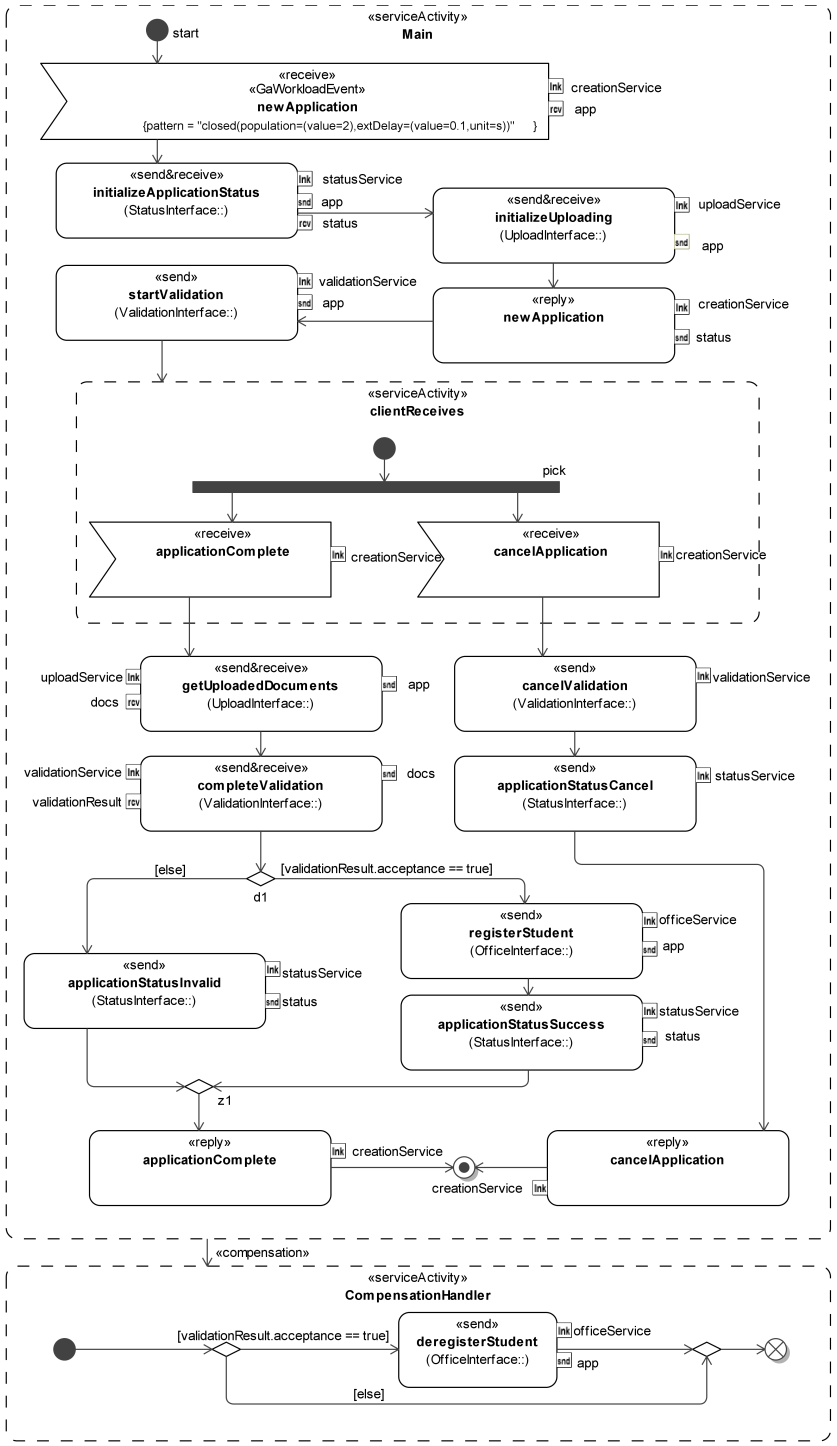


Fig. 8. UML4SOA activity diagram showing the ApplicationCreator.

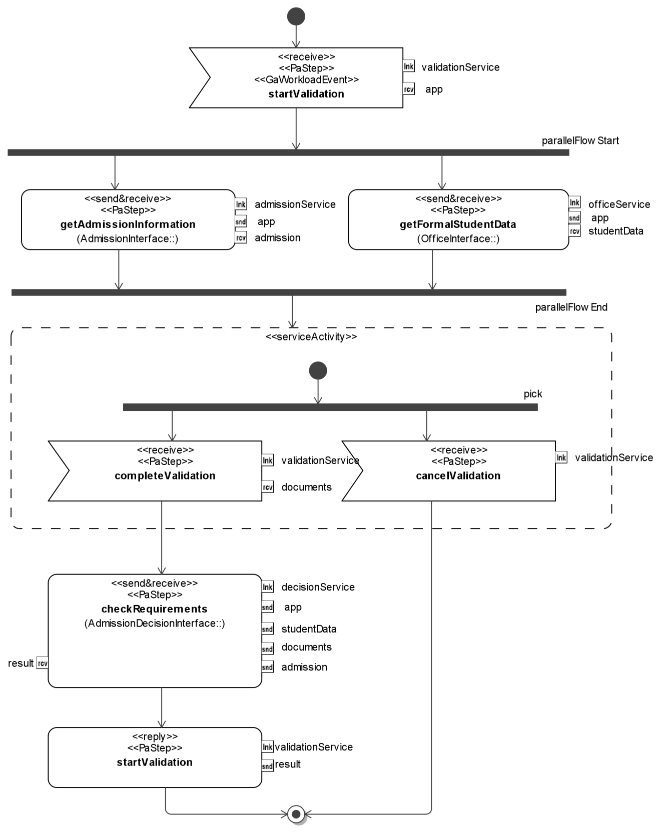


Fig. 9. UML4SOA activity diagram showing the ApplicationValidator.

The second activity diagram, modeling the *ApplicationValidator*, is shown in Fig. 9. This service acts as supplier to the creation service, starting with the receipt of the *startValidation* call from the *ApplicationCreator* through the *validationService* port. Afterwards, both the *OfficeService* and the *Admission-Service* are contacted simultaneously to check admission of the student, and to check the student data. Subsequently, the process waits for the *completeValidation* call from the *ApplicationCreator*. After it is received, all the information gathered so far is checked with the help of the *DecisionService*, and the result is returned to the *ApplicationCreator*.

5 Conclusions

This chapter has discussed three of the scenarios of the Sensoria case studies in detail, presenting graphical models of structure and behavior as well as the implementation of a demonstrator.

Firstly, the *CreditRequest* scenario of the Finance case study has been discussed. This scenario has been used as a test bed for demonstrating the feasibility and effectiveness of the use of the Sensoria process calculi and some of their related analysis techniques and tools. Moreover, it has been used to provide an effective implementation of (part of) the Sensoria approach, specifically modeling and formal analysis of service-oriented software based on mathematically founded techniques. Chapter 7-4 further details the use of formal analysis tools on the credit request scenario.

The Automotive case study scenarios and in particular, the *OnRoadAssistance* scenario, have been widely used by the project partners to illustrate the approaches they implemented with easily understandable examples. The demonstrator explained in this chapter has shown how the Sensoria techniques can be employed for a reaching a fully automated, model-driven approach to SOA development.

Finally, the eUniversity case study has taken concepts from familiar ground for researchers, providing an ideal playground for testing new approaches, methods, and tools for the support of service-oriented architectures. The *StudentApplication* scenario which has been presented in detail in the previous section has been used for qualitative and quantitative analysis integrated with model-driven development and the generation of a running system based on web service technology.

All three case study scenarios are available for download from the Sensoria web site (`http://www.sensoria-ist.eu`).

References

1. M. Alessandrini and D. Dost. Finance Case Study: Requirements, Specification and Modelling of Selected Scenarios (D8.3.a). Technical report, S&N AG, 2007.
2. N. Koch and D. Berndl. Requirements Modelling and Analysis of Selected Scenarios: Automotive Case Study (D8.2.a). Technical report, FAST GmbH, 2007.
3. P. Mayer, A. Schroeder, and N. Koch. MDD4SOA: Model-Driven Service Orchestration. In *The 12th IEEE International EDOC Conference (EDOC 2008)*, pages 203–212, Munich, Germany, 2008. IEEE Computer Society.
4. OMG. Service Oriented Architecture Modelling Language Beta 1. `http://www.soaml.org/`, 2009.
5. R. Xie and N. Koch. Automotive CASE Study: Demonstrator. Technical report, Cirquent GmbH, 2009.