Sensoria

016004

Software Engineering for Service-Oriented Overlay Computers

www.sensoria-ist.eu

D8.8

SENSORIA Results Applied to the Case Studies

Lead contractor for deliverable: ISTI
Author(s): Maurice H. ter Beek (ISTI)
with contributions from L. Acciai, M. Boreale, R. De Nicola, M. Loreti, R. Pugliese, F. Tiezzi (DSIUF), H. Gao, F. Nielson, H.R. Nielson (DTU), H.T. Vieira (FFCUL), S. Gnesi, D. Latella, M. Massink, F. Mazzanti (ISTI), A. Knapp, N. Koch, P. Mayer, M. Wirsing (LMU), C. Bodei, R. Bruni, M.G. Buscemi, G. Ferrari, C. Montangero, L. Semini (PISA), H. Foster (LSS-Imperial), A. Mukhija (LSS-UCL), A. Clark, S. Gilmore, M. Tribastone (UEDIN), L. Bocchi, J.L. Fiadeiro, S. Reiff-Marganiec (ULEICES), C. Guidi (UNIBO), P. Quaglia (UNITN), G. Marczyński, and A. Zawłocki (UWARSAW)

Due date of deliverable: February 28, 2010
Actual submission date: January 13, 2010
Revision: 1
Dissemination level: 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, CIR

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

Information Society
Technologies

Executive Summary

In this deliverable we provide an overview of the application of the results obtained in SENSORIA (i.e., those techniques, methods and languages developed in the technical work packages WP1-WP7) to SENSORIA case studies (developed in work package WP8). As a natural follow-up to deliverables D8.5, D8.6 and D8.7, this deliverable only contains the applications that have been carried out *after* M36.

Contents

Introduction

This deliverable provides an overview of the relations between the SENSORIA case studies from WP8 and the technical work carried out in the work packages WP1-WP7. This deliverable contains those applications that have been carried out *after* M36, making this deliverable a natural follow-up to the M36 deliverables D8.5, D8.6 and D8.7, in which the relations between the case studies and the technical work packages carried out before M36 have been presented—separately for each of the three Themes of SENSORIA. These four deliverables together thus nicely illustrate the central role that the case studies have occupied in *feeding* and *steering* the research in SENSORIA.

This deliverable is structured as follows. After this introduction, we summarize a series of contributions that report on applications of techniques, methods and languages of WP1-WP7 to the case studies.[1] We describe the experience of applying a particular technique, method or language to a case study scenario, but *not* the technique, method or language itself, which are presented in the technical work packages' deliverables.[2] As such, the goal of this deliverable is to answer the following questions:

Aim: besides validating the technique, method or language against the requirements of the case study, was there a specific *aim* that triggered the use of the case study?

Experience: which were the *problems* (if any) faced when applying the technique, method or language to the case study and which are the *results* that have been obtained?

Benefits: what have been the *advantages* (from the engineering, scientific or business point of view) of applying the technique, method or language to the case study?

Feedback: did the application of the technique, method or language to the case study lead to *improvements* of that technique, method or language?

The contributions are organized according to the three themes of SENSORIA.

The first theme deals with linguistic primitives for services and their interaction and composition. These languages are developed on two levels of abstraction: an *architectural* (e.g., UML4SOA, SRML) and a *programming* level (e.g., COWS, Jolie, CaSPiS, CC). This theme constitutes the work packages WP1, WP2 and WP5.

The second theme, constituting work packages WP3 and WP4, deals with type systems, logics, and extensions of the process calculi developed in Theme 1 (e.g., SocL, SoSL, MarCaSPiS, sCOWS) in order to develop verification techniques (e.g., CMC- UMC, PEPA software toolkit) for the *analysis* of behavioural, performance and QoS properties of services.

The third theme, finally, deals with the engineering aspects of services: *model-driven development* (e.g., MDD4SOA, SDE), *deployment* and *reengineering* (e.g., Modes, Dino). This theme constitutes the work packages WP6 and WP7.

Note that for some contributions the choice for the Theme in which to present them was not obvious, because they involve the use of a language from Theme 1 and an analysis technique or tool from Theme 2. Examples include COWS and its model checker CMC (presented in both Themes), sCOWS and its analysis tools sCOWS_LTS and sCOWS_AMC (presented in Theme 1), CaSPiS and its type system and control flow analysis (presented in Theme 1), CC and its model checker ChorSLMC (presented in Theme 1), λ^{req} and its static analysis and model-checking techniques (presented in Theme 2), and CaPiTo and its flow logic analysis (presented in Theme 1).

This deliverable concludes with a synthetic overview of the case studies to which the SENSORIA techniques, methods and languages have been applied, followed by more detailed analytic overviews of the specific experience/benefits of these applications, organized by theme. These overviews include the applications of SENSORIA techniques, methods and languages to the case studies that have been carried out before M36 and that have been discussed in the M36 deliverables D8.5, D8.6 and D8.7.

[1] We include applications to the Bowling Robot demonstration described in Deliverable D8.9.

[2] We refer to the deliverables from the technical work packages for all references to these techniques, methods and languages.

1 Linguistic Primitives

The research in Theme 1 focuses on the development of a generalized concept of service for global computers through the introduction of novel semantically well-founded modelling and programming primitives for services.

1.1 Architectural Level

1.1.1 UML4SOA

The UML2 profile UML4SOA is an implementation- and platform- independent means of modelling service-oriented systems, in particular *service interactions* and *orchestrations*.

Aim: The aim of the application of the UML4SOA modeling technique to the case studies was to prove the usefulness of the UML4SOA profile in practice, and to identify any general problems stumbled upon during this application to improve the UML4SOA profile.

Experience: During the application of UML4SOA to the case studies, we have noted several problems which have led to a new and extended version of UML4SOA which is reported in deliverable D1.4b and in the SENSORIA book chapter "UML Extensions for Service-Oriented Systems". The details are listed below under feedback.

Benefits: UML4SOA is an implementation- and platform-independent means of modelling SOA systems, in particular service orchestrations. UML4SOA diagrams are the basis for various kinds of (formal) analysis tools which can be used for verifying service orchestration behaviour; therefore, by modelling the case studies in UML4SOA, we can take advantage of these verification methods. Furthermore, UML4SOA can also be transformed to platform-specific models like the Web Service Stack (BPEL/WSDL), thus a separate implementation step is not required.

Feedback: Firstly, the need for an explicit specification of the static aspects of a SOA system have been identified. For this, we have included specific support for the soaML profile (created by a task force at OMG), which allows the modeling of services, provided and required ports, interfaces, and message types in addition to the behavioural specifications in UML4SOA. Secondly, we have identified the need for data handling within UML4SOA diagrams, which allow creation, manipulation, and sending/receiving of UML-typed data to and from partners. Finally, the diagrams created for the case studies became quite large, requiring a means to swap out parts of orchestration processes. We have therefore included subscoping in UML4SOA.

1.1.2 SRML

The SENSORIA Reference Modelling Language (SRML) offers primitives for modelling *business services* and *activities*, in which interactions are supported through interfaces. SRML supports a methodological approach that includes the use of the UMC model checker for qualitative analysis and of the Markovian process algebra PEPA for quantitative analysis of timing properties.

Aim: SRML was applied to scenarios from the Finance case study [BFL08, BFG$^+$], in particular to validate the extension of SRML with timing aspects and the use of PEPA together with SRML for the analysis of timing properties.

Experience: The scenarios guided the development and contributed to the validation of several issues:

- a methodology involving UML and SRML: capturing requirements with a SRML profile for use case diagrams, describing the overall interaction with sequence diagrams and describing orchestrations with statechart diagrams;
- define nontrivial SLA properties;
- perform quantitative analysis with PEPA.

Benefits: The advantages of applying SRML to the case study are in the ability that SRML provides to model services (a mortgage brokerage service in particular) at a higher level of abstraction in which the discovery of external services is not programmed as part of the orchestration but instead relies on optimization of quality-of-service constraints expressed declaratively and matchmaking is based on functional properties expressed through behavioural properties again expressed declaratively. A further advantage came through the integration of timing aspects in the language that is used for defining service modules, which then allows modellers to analyze both quantitative and qualitative properties of services over the same specifications.

Feedback: We improved the definition of SLA variables. SLA variables negotiated during the reconfiguration have also been introduced in the behavioural specifications to influence their behaviour. We furthermore defined a prototype for use case diagrams and SRML, and a notation for statechart diagrams that allow one to describe orchestrations that can be directly encoded with PEPA. Its application to the case study led to an improvement of SRML, in the sense that we were able to consolidate the usage of UML-like statecharts for specifying business roles (orchestrations), which led to the definition of a particular profile, and also the method in general through the incorporation of quantitative analysis of timing properties (in addition to the qualitative analysis already defined based on the UMC model checker).

1.1.3 StPowla

The Service-Targeted Policy-Oriented WorkfLow Approach (StPowla) is a workflow-based approach to business process modelling integrating a simple graphical notation to ease the presentation of the core business process, policy language Appel to provide the necessary adaptation to the expectations of the business stakeholders, and the SOA to assemble and orchestrate services in the business process.

Aim: We used the Finance case study to exemplify the StPowla approach. In particular, we have shown how a number of policies can be used together to refine the loan application workflow: the capability to express policies can enhance workflow techniques. The process is rather generic: it shows stages of submission, checking and offer creation and final approval—these stages are essential in the process as specified by the bank to ensure transparency. However, we can imagine a number of refinements that adapt the process to given situations, which we express as policies. We used both the Finance and the Automotive case studies to validate the verification techniques we proposed to address the detection of policy conflicts. The Automotive case study provides a flat domain of physical locations, which has been useful to address the verification of distributed policies.

Experience: In [GMRMS07, tBGMS09] we applied the StPowla approach to a credit request scenario from the Finance case study. The business process core is defined in terms of sequential, parallel and decision-based composition of building blocks called *tasks*. We use the UML4SOA to model the case study workflow. The finer details of the business process are expressed by policies. These can define functional and non- functional requirements of a *task execution*. Once expressed in Appel, policies must be composed, and parallel composition may lead to conflicts. These circumstances arise if the

composed policies overlap, i.e. if there are some states in which more than one of the composed policies is applicable. To help the user avoiding conflicts and undesired overriding, the policy wizard can be extended to integrate the analysis techniques defined in [MRMS07, tBGMS09].

Benefits: StPowla contributes to the engineering of service oriented systems by capturing essential requirements at a business level and allowing the inherent variability in these requirements to be expressed at a similar level of abstraction. Of course it would be possible to write a very detailed business process diagram that encapsulates all the policies, but it is typical that while the essential process remains the same, the policies change. In particular we have already shown specializations depending on branch size or loan amount. We can further imagine policies added "by need" lately, maybe by the final user, and this is where conflict detection proves most useful. Overall policies will hence refine the general process to adapt it to specific environments, but they will also allow adjustments to handle current situations.

Feedback: Specifying the case study in StPowla we have noticed that the approach has several advantages, but a major drawback: often the Business Analyst is not computer literate enough to be confident with the language. As a consequence, we investigated an alternative representation for policies, based on a tabular format. A default table template is automatically derived from the model of the workflow, and can be filled interactively by the Business Analyst. Then, the policy is deployed by sending the corresponding table to the policy server which has a built-in compiler to convert it into its own representation.

1.2 Programming Level

1.2.1 COWS

The Calculus for Orchestration of Web Services (COWS) is a modelling notation for all relevant phases of the life cycle of service-oriented applications (e.g. publication, discovery, SLA negotiation, orchestration). Besides service interactions and compositions, important aspects like *fault* and *compensation handling* can be modelled. Extensions allow timed activities, constraints and stochastic reasoning.

Aim: By means of the credit request scenario from the Finance case study, we aim at showing that some techniques developed for COWS can be successfully used to analyse COWS specifications.

Experience: The credit request scenario lends itself to different kinds of analysis. In [GPT10], we verify *behavioural properties* of the described system formally specified using the service specification language COWS. The properties are expressed by means of SocL, a logic specifically designed to capture distinctive aspects of service-oriented applications, and automatically verified by using CMC, an on-the-fly model checker for SocL formulae. Some examples of properties verified over the system modelling the credit request scenario are listed in Section 2.1.1. Notably, in [BLPT09] we have carried out a similar analysis of an older (and 'deprecated') version of the credit request scenario.

In [GPT10], we check *confidentiality properties* of the same system. Indeed, the type system for COWS introduced in [LPT07] permits expressing and forcing policies regulating the exchange of data among interacting services and ensuring that, in that respect, services do not manifest unexpected behaviours. This enables us to check confidentiality properties. For the credit request scenario, the service programmer can specify policies stating that the customer's personal information and the credit request data cannot become available to unauthorized users. From the credit request service's point of view, the service programmer can require the customer not to pass to other services the offer, which has been specifically computed for the customer demands.

Benefits: Modelling SOC systems and applications by using the formal specification language COWS enables their analysis by means of methods and tools designed for COWS specifications. By applying to

the credit request scenario two different analysis techniques, namely model and type checking, we have verified some properties of interest for the described system.

Feedback: Our experiments witness, on the one hand, that the specific mechanisms and primitives of COWS are very well suitable for encoding service- oriented applications (in particular when they are specified by UML4SOA diagrams) and, on the other hand, that meta-theories, proof techniques and analytical tools developed for 'classical' process calculi, such as model checking and type systems, can be tailored to the needs of service-based applications. This 'proof technology' can eventually pave the way for the development of automatic property validation tools, like the model checker CMC.

Such feedbacks have encouraged us to develop a framework that aims at allowing service designers to specify SOC applications by means of UML4SOA diagrams and, through an automated translation into COWS, to analyse them by means of the COWS- based verification tools. In [GPT10], we present two prototypical software tools implementing the core of this framework.

1.2.2 sCOWS

sCOWS is a stochastic extension of COWS, allowing quantitative analyses with its related tools. In the tradition of stochastic process calculi, the main syntactic difference from COWS is that in sCOWS basic actions are associated with a random variable expressing their rates. Two distinct tools have been developed. One of them, called sCOWS_LTS, allows sCOWS probabilistic model checking through the generation of the LTS (Labelled Transition System) corresponding to the specification, and its subsequent translation to a CTMC (Continuous Time Markov Chain) that can be used as input for the PRISM model checker of CSL (Continuous Stochastic Logic) formulae. The second tool, named sCOWS_AMC, implements approximate statistical model checking of sCOWS terms against CSL. It is a stand-alone tool running a Monte Carlo algorithm and hence based on the generation of simulation traces of the computation rather than on the generation of the global LTS of the specification.

Aim: The credit request scenario of the Finance case study has been modelled in sCOWS to validate both exact and statistical simulation-based model checking of sCOWS against the state-aware logic CSL, and to compare the two kinds of analysis for a realistic case study.

Experience and Benefits: The sCOWS specification of the login phase of the credit request scenario from the Finance case study is described in [CCG$^+$10]. It is implemented at two distinct levels of abstraction. In the first case, referred to as fine-grained, the bank service is composed of a portal service (acting as an interface towards customers) and a login database (which may be thought of as a bank private service). In the second case, called coarse-grained, the login database is not explicitly considered: the whole login process is modelled as if it were completely handled by the portal implementation, and the existence of a login database is abstracted away. Indeed, the two specifications interpret the login process as seen by the bank and the customer, respectively. Both sCOWS_LTS and sCOWS_AMC have been used to analyze the performance of this sCOWS model. In particular, we were interested in checking properties depending on the number of successfully logged on customers.

Examples of properties against which sCOWS models can be checked with sCOWS_LTS are properties like "What is the probability that exactly N customers are logged on at time T?" and "What is the long-run probability of being in a state where exaclty N customers are logged on?".

Despite of the many optimizations implemented in sCOWS_LTS, on some occasions the state space of a specification can be so big that the generation of the CTMC, as well as the application of numerical algorithms to solve the chain, become really demanding tasks. To face this problem and check properties of sCOWS services without generating their complete transition systems, we used sCOWS_AMC. At high level, given a formula φ, its model checking consists of the following steps:

- computation of execution traces obtained by direct simulation of the source sCOWS specification;

- evaluation of φ against each of the traces;
- statistical reasoning based on the number of samples performed and on the desired error threshold.

sCOWS_AMC currently supports the verification of CSL transient time-bounded path formulae like "Is there a probability of at least 80% that in a certain execution time range the number of logged-on customers varies from a value less than two to a value greater or equal two?" and "What is the probability that in a certain execution time range the number of logged-on customers varies from a value less than two to a value greater or equal two?"

As expected, for the coarse-grained specification the simulation- based method is less efficient than the other. This is due to the fact that the state space is not very large, and thus relatively cheap to build. Conversely, when the number and the complexity of the involved services increas, the method based on the generation of the complete LTS gets outperformed by the approach used in sCOWS_AMC.

Feedback: The application to the case study scenario has demonstrated the feasibility of sCOWS_LTS and sCOWS_AMC to analyze the performance of sCOWS models.

1.2.3 Jolie

The Java Orchestration Language Interpreter Engine Jolie has a properly extended semantics that coincides with the Service- Oriented Computing Kernel (SOCK), a calculus that closely follows current technologies with message routing based on *correlation sets* and *request-response service invocations* in WSDL style as primitives.

Aim: We provide a full implementation of the service- oriented architecture of the Finance case study by using our language Jolie. In this way, we intend to show how Jolie can be considered as a mature technology for developing distributed applications by using a service oriented programming paradigm.

Experience: The Finance case study is generally approached by using a standard web solution where a web portal layered on a single database is used. The design of such an application is not obvious from a service-oriented perspective because we need to design it as a composition of services. Thus we require a design phase for modelling services and then implementing it with some kind of technology. The implementation of the case study is available on the web [GM09a].

Benefits: The main advantage of using Jolie is that it fully implements a service-oriented programming paradigm. In Jolie everything is a service. Services can be embedded and aggregated in order to obtain more complex services. Thus, designing a service-oriented architecture in Jolie is simple because the language forces the programmer to develop only services. In particular, a programmer can immediately approach the design of the case study by using a service-oriented programming paradigm.

Feedback: The application allowed us to discover a new kind of service architecture pattern that we call SoS (Service of Services). In a SoS a service can be considered a proprietary resource of a client instead of a simple session. In particular, in the case study we exploit the SoS for associating a specific service to each employee and supervisor. A detailed description of new service-oriented architectural patterns discovered in Jolie can be found in [GM09b].

1.2.4 CaSPiS

CaSPiS is a process calculus for service-oriented applications based on the notions of *sessions* and *pipelines*. A session corresponds to a private channel, instantiated upon service invocation, that binds the caller and the callee. Pipelines are used to manage dataflow among sessions. CaSPiS also provides linguistic primitives for handling programmed session closures.

Aim: We have provided a CaSPiS formalization of a significant portion of the Finance case study. Subsequently both the type system for CaSPiS, developed in [AB08], and the Control Flow Analysis for CaSPiS, developed in [BBB09], have been applied to analyze the resulting model. The aim of analyzing it with the type system for client progress was twofold: demonstrating the expressiveness of CaSPiS and the flexibility of the proposed technique, on a nontrivial system.

The Control Flow Analysis was applied to a specific part of the scenario, illustrative enough to cover the main primitives of CaSPiS, yet small enough to focus the analysis on the relevant details of the interaction. The analysis is able to detect and prevent logic flaws that may allow a user to do something that should not be allowed, just by abusing or misusing the application functionalities. The circumvention or misuse of the required operations can lead to undesired behaviour or to security attacks, called *application logic flaws*. The analysis statically approximates the behaviour of CaSPiS processes, in terms of the possible service and communication synchronizations. The application to the case study shows how this technique could be exploited to address security from the very beginning of the application design.

Experience: Although not supported by an automatic tool, the client progress analysis of the case study was overall quite smooth. It provided us with suggestions for possible extensions of the type system (see last paragraph) and additional insight on the functioning of the Credit Portal scenario. The type system works for a fragment of CaSPiS tailored to *stateless* services: indeed, the considered fragment does not allow return of values on the service side. Moreover, the type system cannot handle the sessions closure primitives. These limitations forced us to consider a simplified, albeit still interesting, version of the case study, where service protocols do not produce effects visible from outside the session. Moreover, closure primitives had to be replaced by simple messages, by which anyone wishing to leave a session communicates this intention to the other party of the session.

The use of CaSPiS, in general, and its session mechanism, in particular, have allowed us to carry the Control Flow Analysis at the right level of abstraction. For instance, it is guaranteed that sibling sessions established between different instances of the same service and the corresponding clients do not interfere one with the other. As a consequence, the analysis can address each client- server conversation separately. Furthermore, it is possible to focus on the application logic, neither having to commit on any specific implementation of sessioning over a certain platform, nor needing to worry about the analysis of such a realization. The Control Flow Analysis has not been implemented yet, but its application to the case study gives some hints on how to extend and adapt the analysis to deal with other security properties and how to exploit possible synergies with the type systems.

Benefits: Applying the type system for client progress to the Credit Request scenario provides with formal guarantee that each session, originated by a service invocation during the evolution of the system, will not block until the client protocol is waiting for either sending data to or receiving data from the service side. This means that, from the client point of view, any interaction with the service will successfully be completed (of course "successfully" does not imply that the requested credit will be ever granted, but just that the client will reach the end of its interaction protocol).

Control Flow Analysis makes it possible to analyze the case study scenario and to detect its logical flaws. The analysis indeed is able to implicitly consider the possible presence of the *malicious customer*. He/she is an *accredited customer of a service*, that has no control of the communication channels, but does not necessarily follow the intended rules of the application protocol and can try to use the functions of the service in an unintended way. Once detected possible misuses, it is also possible to prevent them by instrumenting the code with suitable run-time checks.

Feedback: The experience we gained on the case study by applying the type system for client progress indicates that static analysis techniques can be useful to analyze SOC applications. The results obtained in analyzing the case study by Control Flow Analysis show how to exploit this technique to address security issues in service design, since a service should be efficient and correct when used by an honest

client, but should also be protected by possible malicious clients. The presence of automatic validation tools would be of great help for the designer. Were such a tool developed, then one could proceed by extending the type system to the full calculus, in order to be able to handle both programmed closure of sessions and non-stateless services. It would also be helpful to extend the analysis to deal with the full version of CaSPiS, since at presents it only deals with its `close`-free fragment.

1.2.5 CC

The Conversation Calculus (CC) is a minimal typed model for expressing and analyzing *interaction* in service-oriented systems. It is based on a novel notion of *conversation* which extends the notion of session in a novel direction, allowing, in particular, the specification and analysis of dynamically established service collaborations between multiple parties, which is important to support features such as dynamic discovery of services.

Aim: We applied the CC to the credit request scenario from the Finance case study. The main purpose was to illustrate the applicability of our technique in providing an automated means to assert, given a CC implementation of a system, that such implementation complies to a WS-CDL-like choreography, based on model checking of translations of CC programs and choreographies in pi-calculus and spatial logic specifications, respectively, using the Spatial Logic Model Checker tool [VCV].

Experience: We did not meet any relevant problems in our analysis of the case study, but naturally we had to abstract away from some details and focused solely on the communication model, as could be expected since we are using a minimal process calculus. The main result of this demonstration is the fact that we may assert that the CC implementation of the credit request scenario complies with a well-defined WS-CDL-like choreography, a property which is guaranteed by the automated analysis of the system implementation and choreography [BCL$^+$10].

Benefits and Feedback: As immediate by-products of the application of our technique to the credit request scenario we have obtained a CC model (with a well-defined semantics) of the scenario and a specification of the global interaction scheme (the choreography) of the credit request scenario that may be carried over to other settings, e.g., to check if the implementation meets the expected message sequence charts. Also, the fact that CC specifications may be fed to the Spatial Logic Model Checker allows us to check many other properties of interest (e.g., deadlock freedom).

1.2.6 CaPiTo

CaPiTo is a process calculus for modelling service-oriented applications at both the *abstract* and the *concrete* level, thus achieving a certain level of abstraction without being overwhelmed by the underlying implementation details, but respecting the concrete industrial standards used for implementing the service-oriented applications.

Aim: The objective was to develop a process calculus to model Service-Oriented applications, and a scenario from the Finance case study in particular, so that, on the one hand, we can achieve a certain level of abstraction without being overwhelmed by the underlying implementation details and, on the other hand, respect the concrete industrial standards used for implementing the service-oriented applications. In addition, we applied the Flow Logic approach to the scenario and verified it works as expected.

Experience: The analysis of a credit request scenario from the Finance case study suggests that the authentication property holds. For example, one can safely draw the conclusion that once the decision has been made that whether the request has to be validated by the service for enterprises or corporate, it

cannot be tricked into being processed by the wrong one. Furthermore, it also suggests that no sensitive data is leaked to the attacker, hence confidentiality holds as well. The results are recorded in [GNN09].

Benefits: In our view the main contribution of the CaPiTo approach is that, on the one hand, it allows to perform an abstract modelling of Service-Oriented applications and, on the other hand, facilitates dealing with existing industrial protocols. It is due to this ability that we believe CaPiTo overcomes a shortcoming identified in SENSORIA—there is a gap between the level of models and analyses performed by the academic partners and the realizations and implementations performed by the industrial partners.

Feedback: The process calculus CaPiTo is powerful enough to model not only a credit request scenario from the Finance case study but also Service-Oriented applications in general, in particular when cryptographic protocols are used to ensure security.

1.2.7 SRMC

The SENSORIA Reference Markovian Calculus (SRMC) is a stochastic process calculus which explicitly represents *uncertainties* about system configuration in addition to the controlled randomness of an underlying stochastic process. SRMC can be seen as an extension of PEPA (formally, a superset). The kinds of analysis that can be performed on SRMC models are supported by a suite of software tools that provides tight integration with the PEPA modelling and analysis tools.

Aim: A significant concern with the analysis of SOC is the treatment of *uncertainty* about service invocation. Services are replicated across the network and different service instances have different performance characteristics due to the inherent heterogeneity of large-scale distributed systems. Further, it is not appropriate to assume that all of these instances of the service are functionally identical because some might be running the latest version of the service and others might be running an older version. Different policies at different sites might cause features to be turned off, for security or other reasons. SRMC was developed to face these challenges and tested by applying it to the eUniversity case study.

Experience: We addressed the challenge of evaluating service-level agreements in the presence of this kind of uncertainty [CGT08] and created a novel analysis approach which allows precise quantitative statements to be made about such systems. We tested the expressivity of the SRMC modelling language with a case study of a Web-service orchestration [CGT09a]. We further used the language to determine the scalability of the eUniversity system in the presence of increasing numbers of student users and uncertainty about system configuration [CGT09b].

Benefits and Feedback: A novel analysis approach was created which allows precise quantitative statements to be made about service-level agreements in systems in the presence of uncertainty.

1.2.8 cc-pi

The cc-pi calculus is a constraint-based language that supports dynamic selection of services by allowing to specify negotiations among service providers and requesters on the QoS. The cc-pi calculus combines basic operations of concurrent constraint programming with a symmetric, synchronous mechanism of interaction à la pi-calculus. [BM10] presents a variant of the cc-pi calculus that features a choice operator whose branches have a dynamic form of priority and reports on the application of such a prioritized calculus to the credit request scenario from the Finance case study.

Aim: We have employed the Finance case study for validating the main features of the prioritized cc-pi calculus. In particular, we have shown the benefits of the novel prioritized choice by specifying the negotiation protocol followed in the credit request scenario involving partners who have a given order of preference between their possible alternatives.

Experience: We have focused on the QoS agreements reached in the credit request scenario. Specifically, we have formalized in the prioritized cc-pi calculus two QoS negotiations that depend on each other. The first negotiation is between a customer requesting a mortgage and the bank and concerns the *response time* of the credit request service. The second negotiation is not only on the response time but also on the cost and involves the bank which in turn invokes a financial service (for obtaining a customer profile) and the provider of such a service. The prioritized cc-pi allows modelling the fact that the bank initially offers a given price for the financial service and, in case no service is found, offers a higher price for the same service until either an agreement is reached or a maximum price threshold is exceeded.

Benefits: The QoS requirements and guarantees are specified in cc-pi in terms of constraints and a QoS negotiation between two or more parties succeeds if the store of constraints resulting from the combination of the respective constraints is consistent. Hence, checking whether a contract is reached amounts to verifying consistency over a given constraint system, thus possibly exploiting existing tools/techniques. Furthermore, cc- pi is parametric with respect to the choice of the constraint system. Hence, by varying its underlying semiring structure it is also possible to model soft constraints which can represent not only boolean values but more informative values instead.

The use of constraints also allows modelling the fact that two or more negotiations can be related. For instance, in the cc-pi specification of the credit request scenario, in order to be able to respect the contract with the customer, the bank requests to the financial service the same response time that it has to ensure to the customer for the credit service.

Feedback: The experiments over the Finance case study showed that the prioritized cc-pi can be effectively employed to specify complex QoS negotiations in which partners have preferences over their possible moves. Our approach would benefit from an implementation of the transition system of the calculus, which would provide the designer an automated tool for modelling and checking QoS contracts.

1.2.9 Institutions

A logical specification framework based on *institutions* [BG92] allows for declarative specifications and modelling of SOAs.

Aim: Declarative software systems specification techniques provide abstract logical description of the system as a whole. We aimed at defining a logical specification framework, preferably based on institutions, that would provide means for declarative specifications and modelling of SOAs. By using the course selection scenario from the eUniversity case study scenario, we wanted to unveil the details of our logics and show the appropriate levels of description of SOA system properties.

Experience: We have proposed a heterogeneous approach to SOA system specification [KMWZ10]. It consisted of institutions suitable for specifying SOA systems in which services are classified by roles (or interfaces), the interaction structure of services may change over time and the number of services may be not known in advance. The course selection scenario from the eUniversity case study gave us the unvaluable reference that kept our formalism as close as possible to the SOA paradigm. The scenario has showed that it was useful to separate the description of the behaviour of individual services from their collaboration. Thus, we have proposed two different logical systems formalised as institutions. In the local logic one could specify the behaviour of individual services, whereas the global logic was to specify

the choreography of a SOA system, i.e. the common behaviour of cooperating services. It provided means to refer to individual system components as well as to describe the overall system configuration and its evolution in time.

Benefits: To model changing interaction topology present in the case study scenario, we have used a synchronisation predicate $\sim$ in our global logic for modelling the communication connections of the service partners; this communication structure could have changed dynamically (i.e., it may have been different for each configuration). The case study scenario contained unspecified number of service providers. As a consequence, our global logic contained quantification over components which has allowed us to write specifications for systems with an unbounded number of services.

Feedback: We have used the course selection scenario for constant validation of our approach. It has proved that the two logical systems that we defined, the local logic to describe the individual services and the global logic to describe the inter- service interactions, are sufficient to model real-life SOA systems.

2 Qualitative and Quantitative Analysis

The research in Theme 2 focuses on the development of mathematical analysis and verification techniques and tools for system behaviour and QoS properties.

2.1 Qualitative Analysis

2.1.1 CMC-UMC

CMC and UMC are two prototypical instantiations of a common logical verification framework for the analysis of *functional* properties of service-oriented systems. They only differ w.r.t. the underlying computational models, which are built from COWS specifications in the case of CMC and from UML statecharts in the case of UMC.

Aim: After having applied the CMC-UMC framework to the Automotive and Telecommunications case studies, it has recently been used to analyse the credit request scenario from the Finance case study and to analyse the Bowling Robot case study [GPT10, tBBG09, tBLLP10, tBM10].

Experience: Our applications consisted of model checking behavioural properties expressed in SocL, a logic specifically designed to capture distinctive aspects of service- oriented applications, and for which a set of patterns of service properties was defined.

Some examples of properties verified over the system modelling the credit request scenario are:

- the credit request service is *available*, i.e. it is always capable to accept a login request;
- the service is *responsive*, i.e. it always guarantees an answer (an offer or a decline) to each received credit request, unless the customer cancels his own request;
- the service can accept a cancellation of a credit request until the customer effectively requires it or receives an answer;
- if a cancellation is required after the rating calculation is started, all compensation activities will be executed.

Benefits: Our verification experiences allow us to conclude that certain intended properties of the case study scenarios are indeed valid and, moreover, to show the usefulness and feasibility of a formal approach to specify and rigorously analyse system designs, also in industrial contexts.

Feedback: The application of CMC-UMC to the various case studies has led to a fine-tuning of both the underlying SocL logic and the model-checking approach. The obtained feedback has moreover steered the development of frameworks allowing service designers to specify service-oriented applications in UML4SOA diagrams and, through an automated translation into either COWS or UML statecharts, to analyse them by means of the CMC-UMC verification framework.

2.1.2 ESC

Event-based Service Coordination (ESC) is a middleware supporting the design and implementation of service coordination policies (both *orchestration* and *choreography*). The middleware consists of a set of API for assembling services by exploiting the *event notification* paradigm. A distinguishing feature of ESC is the facility to manage *long running transactions*. At the abstract level, the middleware takes the form of the Signal Calculus (SC), an asynchronous process calculus where service interactions are managed by issuing and reacting to suitable (multicast) events. Remarkably, the middleware does not assume any centralized mechanism for publishing, subscribing and notifying events. Hence, SC and ESC have to be properly regarded as a foundational framework and its programming counterpart for specifying, verifying and programming coordination policies of distributed services.

Aim: We aim at experimenting the ESC framework in the design, implemention and verification of properties of services coordination policies (orchestration and choreography). Our main goal is to show that the formal mechanisms underlying the ESC framework allow a more robust development of coordination policies for services.

Experience: To illustrate some of the features and design facilities made available by the ESC framework, we have applied it to the Finance case study [Str09, Gua09]. The focus of our experiments is on the design and implementation of the workflow of the coordination, taking into account the possibility of handling long-running transactions (LRTs) in the style of SAGA compensations.

Benefits: The main benefits of the approach can be summarized as follows

- *The integration of semantic-based verification techniques to manage properties of LRTs*. Our verification technique is based on the notion of *bisimulation*, an observational equivalence, that not only represents the behaviour of sets of components interacting with each other but also that of isolated subsystems. Bisimilarity allows us to distinguish isolated components that behave differently when 'plugged' into different service networks. We formally proved that the constraints on transactional isolation are maintained in the involved components. The verification of LRTs of the Finance case study has been done by checking that it is bisimilar to a 'magic' property, i.e. an abstracted design that models properties of interest.

- The ESC design of LRTs can be automatically compiled into executable Java programs.

Feedback: The application of the methodology to the Finance case study has allowed us to validate our *model- driven development* methodology and the *refactoring* rules for LRTs. Indeed, as a matter of fact, LRTs designs (i) neglect distribution aspects of the transactional activities, (ii) does not specify if activities are atomic or consisting of hidden sub-activities, (iii) delegate activities or compensations. Our refactoring rules address these crucial issues of the deployment phase. Arguably, refactoring does not have to alter the high level meaning of the designer since refactoring rules preserve the intended semantics. Our refactoring rules are proved sound by showing that they preserve (weak) bisimularity.

2.2 Quantitative Analysis

2.2.1 SoSL

The Service-oriented Stochastic Logic (SoSL) was designed for dealing with specific SOC features.

Aim: SoSL has been applied to the MarCaSPiS model of the credit request scenario from the Finance case study. The aim of this application was to experiment with automatic analysis of quantitative properties of the credit request scenario from the Finance case study using the stochastic process language MarCaSPiS as modeling language, stochastic SoSL as property specification language and a prototype stochastic model-checking tool which uses MRMC as a kernel engine. The application is described in some detail in deliverable D4.a.

Experience: In the application three specific aspects of the system modelling the credit request scenario have been addressed: system performance, supervisor and employee workload, and system reactivity. The study has been performed under different sets of assumptions on client ratings. The probability for clients to get served as a function of (waiting) time has been studied and it has been found that such a probability dramatically decreases when the number of clients increases, in particular when there is a large percentage of low-rate clients (50%) relative to the percentage of high-rate clients (17%). Supervisor and employee workload has been found rather low, showing that the service is rather under-used.

Benefits: The use of SoSL for the characterization of properties over the MarCaSPiS model allowed for a simple and concise formalization of aspects (e.g. system performance, workload, and reactivity) and for efficient computation of the above measures, in different scenarios, by means of stochastic model checking. In particular, the open-endedness features of SoSL allowed for the characterization and measurement of system reactivity, i.e. the capacity of the system to react to external requests, where by external it is meant that such requests, and related interactions, did not need to be included in the MarCaSPiS model but have been captured by the related SoSL formulae

Feedback: The example suffered of state-explosion; consequently only scenarios with a few clients active at the same time could be analyzed. Possible ways out are the use of MarCaSPiS discrete simulation or Ordinary Differential Equations semantics, and related solution software tools.

2.2.2 PEPA toolkit

Software tools for the stochastic process algebra PEPA allow for *safety*, *response-time*, *passage-end*, and *sensitivity* analyses. Passage-end analysis is an extension of passage-time analysis, a well-known technique for analysing continuous time Markov chains. Sensitivity analysis is based on using a distributing computing platform to run an analysis many times in parallel, each run representing a different configuration of the rates of activities in the model (i.e. the results are used to reason about the effect a given activity's rate has on the probability of completing the passage). Response-time analysis considers the timed behaviour of a system in the context of a particular workload and a particular sequence of activities that must occur. The analysis determines the probability of completing the work from the start to the stop activity, via any possible path through the system behaviour. This probability is then plotted against time.

Aim: The e-University case study was used as a framework to illustrate the main theoretical developments of PEPA w.r.t. its deterministic interpretation. Moreover, *response-time* and *sensitivity* analyses have been applied to the credit request scenario from the Finance case study [CCG+10], with the intention of identifying the bottleneck activity.

Experience: A noteworthy contribution of [TG10] is a general modelling strategy which may be applied to a variety of distributed applications, including service-oriented systems. The performance model consists of a workload component which makes access to the eUniversity application, modelled as pool of execution threads, each for a specific kind of service provided by the system. In turn, a thread is run on a processor component, which ultimately determines the processing capacity of the system.

Sensitivity analysis was used to generate a family of cumulative distribution functions detailing the response-time distribution of the credit request process, which identified decision making as the bottleneck activity. This quantitative analysis was technically particularly challenging because it is a *multi-scale* problem where rates are separated by several orders of magnitude and the attendant numerical problem is *stiff* (i.e., computationally expensive).

Benefits: The application to the eUniversity case study has revealed that one of the key benefits of PEPA is its capability of targeting both a stochastic and a deterministic semantics. Each has its own strengths and weaknesses. The stochastic interpretation may be very accurate, but is fundamentally limited by a very rapid growth of the reachable state space of the system, which makes the analysis unfeasible even for relatively small systems. However, this discrete-state representation is still a very useful tool for the purposes of model debugging. With the aid of the graphical software toolkit for PEPA, the state space may be interactively walked through to ensure that the model matches the intended behaviour (e.g., no thread starts autonomously, but its execution is triggered by a user request). With the deterministic semantics, the fine granularity of the discrete-state representation is lost in favour of a collective view of the system's dynamics, which results in a highly scalable mathematics for performance evaluation. Interpreting the PEPA model as a set of ordinary differential equations permitted the analysis of very large eUniversity systems consisting of thousands of workload components (which should have been alternatively described by billions of states). As an application, we presented a simple optimization study aimed at minimizing the system resources while maintaining satisfactory levels of performance.

The analyses of the credit request scenario to investigate which is the bottleneck activity have allowed us to decide where effort would be best spent in improving functions within the entire business process.

Feedback: The analyses of the case studies have prompted further work, particularly in the area of tool development. The PEPA toolkit has been recently enriched with a collection of modules that help specify the desired performance metrics of a model, shielding the user from the direct inspection of the solution of the differential equation, which may often be too raw to gain insight into the system behaviour.

2.2.3 λ^{req}

λ^{req} is a core functional calculus for services and service orchestration, featuring primitives for selecting and invoking services that respect given behavioural requirements (*call-by-contract*). Services are modelled as functions with side effects representing the action of accessing *security-critical* resources. Security policies are arbitrary safety properties on program executions. A key point is that security policies are applied within a given scope, so we called them *local policies*. λ^{req} is equipped with a formal methodology that makes secure orchestration feasible. An abstraction of the program behaviour is first extracted, through a type and effect system. This abstract behaviour over-approximates the possible run-time histories of all services involved in an orchestration. The abstract behaviour is then model checked to construct the skeleton structure of the orchestrator that securely coordinates the running services.

Aim: The notion of call-by-contract calls for a matchmaking algorithm based on formal semantic abstractions. To this aim, we have developed a static analysis technique to detect the viable strategies to solve the call-by-contracts involved in a service orchestration. Our static machinery exploits model-checking techniques to determine the orchestration plans driving the selection of those services that match the security contracts on demand. A plan formalises how a call-by-contract invocation is resolved

into the actual service binding. In other words, our planning strategy is a semantic-based method to identify which services the orchestrator must choose to guarantee security of the overall application.

Experience: To illustrate some of the features and design facilities made available by our framework, we have applied it to the Finance case study. The main focus has been placed on the design of resource usage policies [BDFZ09].

Benefits: The proposed methodology features dynamic and static semantics, allowing for formal reasoning about systems. Static analysis and model checking techniques provide the designer with useful information to assess and fix possible vulnerabilities. The main benefits of the approach are as follows:

- *The integration of a verification technique to study the composition properties of service networks*. A static analysis is used to infer an abstraction of the behaviour of a network. This abstraction is then model checked to construct a correct orchestrator that coordinates the services by respecting the policies on demand. The resulting orchestration will also allow for improving the overall performance by avoiding unnecessary dynamic checks enforcing the usage policies of resources while executing services. Studying the output of the model checker may highlight possible design flaws and suggest how the calls by contract and the policies can be revised. All of the above machinery is completely mechanizable and we implemented a tool to support our methodology [BZ09, BDFZ08]. The tool is based on strong theoretical grounds, which positively impacts the reliability of our approach.
- *A study of various planning and recovering strategies*. We discuss several situations in which one needs to take a decision before proceeding with the execution. For instance, when a planned service disappears unexpectedly, one can choose to replan, i.e. to adapt the current plan to the new network configuration. Depending on the boundary conditions and on past experience, one can choose among different tactics. We discuss the feasibility, advantages, and costs of each of them.

Feedback: The application of the methodology to the Finance case study allowed us to properly identify and validate a variety of resource usage policies as well as to experiment our model checker.

3 Deployment and Development

The research in Theme 3 focuses on model-based development techniques for refining and transforming service specifications, novel techniques for deploying service descriptions on different global computers, methods for reengineering legacy systems into service-oriented ones, and a software development process for service-oriented systems.

3.1 Deployment and Reengineering

3.1.1 Modes

The Software Architecture Modes for Service Modes presented in deliverable D8.7 has been extended by providing a modelling approach with formal models and verification techniques [Fos09].

Aim: The technique aims to uphold the principles of *quiescence* to dynamic and adaptive architecture models by verifying different levels of mode model abstraction in architecture and behaviour. The techniques for Service Modes were provided as a solution to various scenarios from the Automotive case study. Modes fit more naturally in these scenarios as Mode architectures themselves are based upon an evolutionary, scenario-based, specification methodology [HKMU06].

Experience: The technique consists of three parts. Firstly a Service Mode model is specified using a UML Modes Profile. The profile defines the scenarios of a service system by identifying particular modes of operation. Secondly, the model is transformed into formal architecture models in the Darwin ADL and formal behaviour models in the Finite State Process (FSP) notation. Thirdly, several types of formal model analysis are conducted. These effectively analyze lower-level to higher-level aspects of a service modes architecture to achieve *quiescence* but others will be considered in future work. *Service Protocol Compatibility* analyzes the expected provided and required service interactions between services in each mode composition. This includes both the ordering of interactions in which services expect their methods to be called, but also the order in which they call other service's methods. The process of analysis takes as input a combined mode architecture and behaviour model, and analyzes each mode service protocol with that of the other services in the mode. *Mode Behaviour Reachability* analyzes the expected service composition behaviour with that which is offered by the services in the mode configuration. The goal is to take the behaviour model of the System and compare it to that of a Mode Architecture (on analysis of the combined System and Architecture model). *Modes Composition Analysis* analyzes the composition of system behaviour specified in all the modes given in the Modes architecture specification. The goal is to take each service mode behaviour specified and consider the events which designate a mode change. Further details of the technique can be found in deliverable D6.1.d.

Modes present a natural way to express different system scenarios. It was evident from the scenarios of the Automotive case study that scenarios of a service system (i.e. the In-Vehicle Services Architecture) needed to be adaptive and dynamic in service provision. Using the requirements of the case study provided a platform to explore how Software Architecture Modes could be used to specify and analyze such requirements. An example Composite Structure Diagram for the on road assistance scenario is presented in Fig. 1. Service Modes extend Software Architecture Modes with mode and system behaviour, service constraints (in the form of Quality-of-Service attributes) and service brokering specifications.

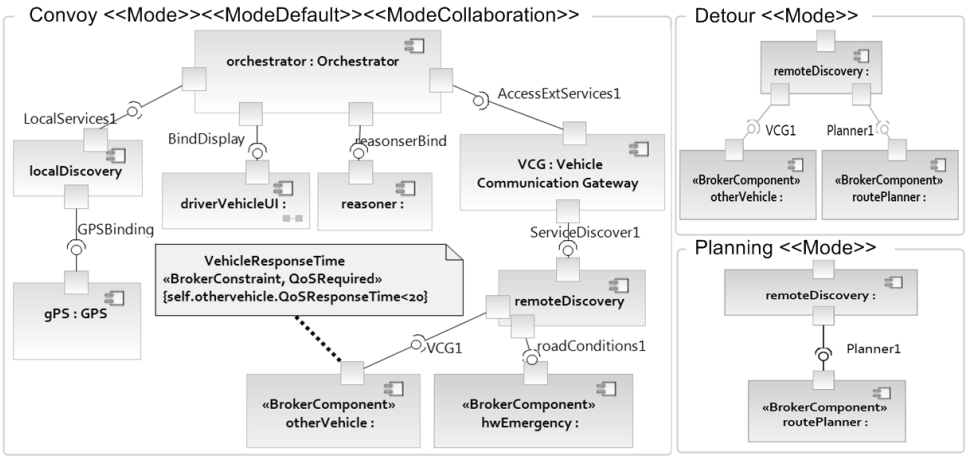

Figure 1: On road assistance planning scenario with Modes Meta-Model

A focus on techniques for analysis of service Modes became apparent when considering how the mode changes occurred through both mode and system behaviour adaptation. For example, an event received by the composition activity for a *Highway Emergency* could eventually lead to a notification of architectural change from Convoy to Detour Mode configuration (illustrated as an activity diagram in Figure 2). Using analysis through model checking the service engineer can check whether the behaviour specified in the Convoy mode (and its services) is compatible for this to happen. At runtime it would be expected that a coordinator agent manages the events and runtime architecture changes (e.g. swapping in and out different service composition processes).

Benefits: Designing service configurations for a SOA has typically been discussed with a static view, in that service compositions and their bindings are designed for a single architecture configuration that

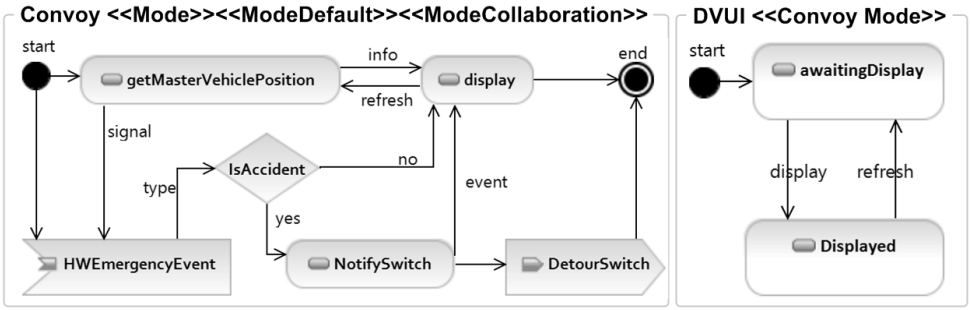

Figure 2: Convoy Service Mode Behaviour specified in Activity Diagram

needs to be modified as new requirements are introduced. This leads to a complex situation of specifying and managing the dynamic requirements of services (both in service request and provision) and how these can be specified in a single architecture model. Adaptive service brokering requirements are typically drawn from both functional and non- functional aspects of services. To assist service engineers in engineering solutions for dynamic service brokering, we have created some steps of an approach to enable modelling, model transformation, broker configuration and runtime support for the tasks involved. In the enhanced approach, engineers can verify that mode elements and compositions are safe and correct using both architecture and behaviour analysis techniques. When analysis proves successful, the generated mode elements can be used to further generate inputs to Dino service brokers (see below) in the form of requirements and capability documents.

Feedback: The result of using Modes to specify, analyze and generate deployment artefacts provides contributions to the areas of service reconfiguration, adaptation and deployment. The approach also provides an end-to-end example from models to runtime environments. Although not specifically stated in the requirements for the Automotive case study, the scenarios discussed in deliverable D8.0 should not be considered mutually exclusive. They will most likely in fact be part of a single service system where different scenarios are carried out in demand by either the driver or the vehicle behaviour. In this way the scenarios can be constructed individually yet considered together to ensure that, once deployed, the system operates in a safe and correct manner. The Modes approach and techniques facilitates service engineers in these difficult and complex engineering tasks.

3.1.2 Dino

The Dino approach provides runtime support for dynamic and adaptive *service composition*. It does so for all stages of a service composition life cycle: service discovery, selection, binding, delivery, monitoring and adaptation.

Aim: The aims of applying Dino to a scenario from the Automotive case study were: firstly, to elicit and analyze the requirements for the Dino brokers with respect to composition of services in an open dynamic environment, and secondly to validate that the resulting design of the Dino brokers meets these requirements satisfactorily. In short, the Automotive case study has provided the requirements, guided the design, and helped to validate the work done in the Dino project.

Experience: The design of the Dino brokers meets the requirements of the specific scenarios from the Automotive case study, which were used for eliciting the requirements for the Dino brokers and for guiding the design of the Dino brokers. These example scenarios and the results are discussed in the above deliverables D6.1.d and D6.3.d and the SENSORIA book chapter “Runtime Support for Dynamic and Adaptive Service Composition”.

Benefits: The Automotive case study has been very useful in guiding the design of the Dino brokers. The concrete requirements provided by the specific scenarios from the case study have guided the design, and have helped to validate the design. The Dino brokers are able to perform dynamic and adaptive composition of services in different kinds of open dynamic environments. The Automotive case study has served as an example of such an environment, and has therefore presented us with concrete requirements to guide the design and validate the same.

Feedback: The use of the Automotive case study has helped to guide the design of the Dino brokers, by presenting the concrete requirements from specific scenarios. Moreover, the case study has allowed the design to be validated and improved based on its application to specific scenarios from the case study. The Dino runtime was extended by S&N with an "intelligent" mode, allowing operators to select their priority of matching services, and integrated into the credit request scenario from the Finance case study [Ale09].

3.2 Model-driven Development

3.2.1 MDD4SOA

The Model-Driven Development Approach for SOA (MDD4SOA) is a tool to *transform* UML4SOA orchestration models to abstract code in executable languages (BPEL/WSDL, Java and Jolie). MDD4SOA is based on model-to-model and model-to-code transformations for *model refinement* and *code generation*, in particular PIM-to-PIM and PIM-to-PSM transformations (where PIM stands for Platform Independent Model and PSM for Platform Specific Model). The particular PIM that is used to transform a UML4SOA model to a PSM is called the Intermediate Orchestration Model (IOM).

Aim: The aim of the application of the MDD4SOA transformer suite to the case studies was to prove the usefulness of the MDD4SOA transformers in practice. In particular, the idea was to test a complete transformation from an UML4SOA model down to actual code and its execution on a target platform, and to identify any general problems stumbled upon during this application to improve the transformers.

Experience: We have applied the MDD4SOA transformers in combination with UML4SOA, and have worked on both areas simultaneously. As UML4SOA changed, we also needed to adapt MDD4SOA to reflect the new features of UML4SOA (static structure, data handling, and subscoping). With regard to MDD4SOA in itself, we have noted several areas of improvement, in particular regarding user feedback, where we display a dedicated dialog with error and warning information which greatly helps the user in finding problems in the original UML4SOA model. Some of these issues are listed in deliverable D1.4b and in the SENSORIA book chapter "UML Extensions for Service- Oriented Systems" alongside UML4SOA. The details are listed below under feedback.

Benefits: MDD4SOA implements the ideas of the model-driven architecture (MDA) for the platform-independent UML4SOA models (PIMs). We allow modellers to automatically transform these models to lower-level, platform-dependent models (PSMs) and source code: Firstly, the Web Service Stack (BPEL, WSDL, XSD); secondly, the object oriented programming language Java, and thirdly the service programming language Jolie. Enabling MDA on UML4SOA greatly decreases the time needed to implement the UML models, thereby saving cost in system implementation and reducing mistakes due to error-prone and repetitive implementation of the information in the models.

Feedback: As mentioned above, MDD4SOA is tightly integrated with UML4SOA, therefore, during the course of the last year we have implemented the new features of UML4SOA, which are the static

modeling of a system, data handling in orchestrations, and subscoping. Furthermore, we have identified the need for a more readable and precise error reporting, which have implemented throughout the transformer, which now informs the user about the precise locations of errors in the source model.

3.2.2 SDE

The SENSORIA Development Environment (SDE) is a *tool integration platform* for the different tools developed in SENSORIA, which allows developers to *find*, *use* and *combine* them. To integrate with existing tools and platforms for the development of SOA systems, the SDE is based on the industry-standard Eclipse platform and its underlying, service-oriented OSGi framework.

Aim: The orchestration features within the SDE have been employed for combining several integrated tools for the development and analysis of the case studies.

Experience: In [XK09], a detailed documentation of the implemented Automotive Demonstrator illustrates the use of a set of techniques, methods and tools from the SDE developed within the scope of SENSORIA (e.g. UML4SOA, MDD4SOA, Dino). The use of the SDE for analyzing orchestrations in the eUniversity case study with the help of three integrated tools (WS-Engineer, PEPA, and MDD4SOA) is described in detail in [MJFT08].

Benefits: The main benefit of the SDE is its integrative nature: all SENSORIA tools are available within the SDE with a uniform API description and the ability to take part in larger orchestrations. Creating orchestrations is possible using either a textual, JavaScript-based approach or a graphical, UML-activity- diagram-like workflow approach. The opportunity to integrate tools into the SDE has prompted SENSORIA tool developers to spend time on thinking about the collaboration of individual tools, which benefits the end user.

Feedback: Feedback from applying tools to the case studies, with the SDE as underlying platform for tool invocation, has led to improved SDE interfaces. The resulting wizard-based service call infrastructure has the advantage of always presenting the same interface to developers regardless of the tool or orchestration. The graphical orchestration mechanism is also a direct result of these applications.

4 Conclusion

In Table 1, we provide a synthetic overview of the case studies to which the SENSORIA techniques, methods and languages have been applied, including applications that have been carried out before M36. This table, organized by theme, clearly illustrates the central role of the *industrial* case studies from the Automotive and—in particular—Finance domains, as well as the more specific role of the *academic* eUniversity case study. Note that only few SENSORIA techniques, methods and languages have been applied to the Bowling Robot and Telecommunications case studies. This is due to the following reasons. First, the *industrial* Telecommunications case study has suffered from the fact that Telecom Italia has considerably reduced its effort in SENSORIA rather early into the project. Second, the Bowling Robot *demonstration* has been introduced late into the project, initially as a game scenario to show SENSORIA's software engineering approach at hands-on demonstrations, after which it has evolved into a case study.

In Tables 2-4, organized by theme, we provide detailed analytic overviews of the specific experience/ benefits of having applied the SENSORIA techniques, methods and languages to the case studies.

Table 1: Validation of SENSORIA techniques, methods and languages.

		Automotive	Finance	eUniversity	Bowling Robot	Telco
T	UML4SOA	√	√	√	√	√
	SRML	√	√			√
H	StPowla	√	√			√
	(s)COWS	√	√			
E	SOCK/Jolie	√	√			
	(Mar)CaSPiS	√	√		√	
M	CC		√			
	λ^{req}	√	√			
E	CaPiTo		√			
	SRMC			√		
1	cc-pi		√			√
	ADR	√				
	Institutions			√		
	SocL	√	√			
T	CMC-UMC	√	√		√	√
H	chorSLMC		√			
E	Flow logic		√			
M	SC-ESC	√	√			
E	OCPR	√	√			
	WS-Engineer	√	√	√	√	
2	SoSL/MoSL	√	√		√	
	PEPA toolkit	√	√	√	√	
	Modes	√				
T	Dino	√	√			
H	JCaSPiS		√			
E	MDD4SOA	√	√	√		√
M	VIATRA2	√	√			
E	GT		√			
	MBTE		√			
3	Patterns	√	√	√		
	SDE	√	√	√		

Table 2: Theme 1: experience/benefits of SENSORIA techniques, methods & languages.

Theme 1	Automotive	Finance	eUniversity	Bowling Robot	Telco
UML4SOA	Test usefulness in practice; Provide models for formal verification				
SRML	Validate primitives, 3-layer approach & definition SLAs; Validate timing extension				Test interactions: add key parameters
StPowla		Validate approach; Now derives policy templates			Assess impact on business process design
COWS sCOWS	Feasibility mechanisms + primitives to model SOA & provide stochastic description				
SOCK	Validate primitives & faults + compensations modelling; Verify dynamic handler & automatic fault notification; Now improved handlers				
Jolie	Test programming a SOA; Found new SOA patterns				
CaSPiS MarCaSPiS	Test its effectiveness & its Markovian extension			Feasibility of methodology	
CC		Type to verify key properties			
λ^{req}	Call-by-contract invocation				
CaPiTo		Model SOA especially if crypto protocols for security			
SRMC			Test expressivity; New analysis approach		
cc-pi		Validate prioritized variant			Validate basic primitives
ADR	Disambiguate informal SOA specifications by formal model of reconfigurations & constraints				
Institutions			Validate approach; Useful to separate behavioural description of services from choreography		

Table 3: Theme 2: experience/benefits of SENSORIA techniques, methods & languages.

Theme 2	Automotive	Finance	eUniversity	Bowling Robot	Telco
StPowla	Detection of conflicts by theorem proving; Needed extension Appel semantics	Detection of conflicts by UMC; Needed definition correspondence Appel–UML			
SocL	Test expressivity for SOA properties; Defined patterns of service properties				
CMC-UMC	Test & fine-tune model checker; Verify properties; Feasibility type checking with CMC; Now automatic translations from UML4SOA to COWS & UMC			Test & fine-tune model checker; Verify properties; Now automatic translations from UML4SOA to UMC	
CaSPiS		Type system to check progress property; Control flow analysis to detect & prevent misuses			
chorSLMC		Verify multiparty interaction; Proved protocols of interaction well-defined + deadlock-free			
SC-ESC	Experiment, evaluate & reason about long running transactions				
Flow logic		Proved authenticity & confidentiality			
OCPR	Defined compositional notion of service equivalence; Verified service equivalence & replaceability				
WS-Engineer	Analyzed correctness & consistency of service compositions; Helped to further develop the tool		Check interactions between orchestrations for deadlocks	Test & further develop the tool	

continued on next page...

Table 3: **– continued from previous page**

Theme 2	Automotive	Finance	eUniversity	Bowling Robot	Telco
SoSL/MoSL	Test expressivity for SOC features; Verify dependability (workload, reactivity) & performance properties of services: difficult & error- prone in natural language			Show methodology provides intuitive & effective estimations of robot behaviour	
λ^{req}	Exploit output of model checker to highlight design flaws, suggesting how to revise the orchestration & security policies				
PEPA toolkit	Passage-end, response-time & safety analysis; Development environment now more user-friendly	Sensitivity & response-time analysis to identify bottleneck activity; technically particularly challenging	Show main developments wrt deterministic semantics PEPA; Its key benefit: target a deterministic & a stochastic semantics	Passage-end analysis: precisely quantified probability of expected outcome	
SRMC			Test software tool suite; Verify scalability in presence uncertainties		
sCOWS_LTS sCOWS_AMC		Analysis of system performance			

Table 4: Theme 3: experience/benefits of SENSORIA techniques, methods & languages.

Theme 3	Automotive	Finance	eUniversity	Bowling Robot	Telco
Modes	Test modelling orchestration & choreography requirements; Created UML2 profile; Extended analysis & WS-Engineer				
Dino	Provide requirements, guide design & validate Dino; Now improved & extended design				
JCaSPiS		Implement a real scenario; Preserve analysis from modelling to implementation level; Now new functionalities			
MDD4SOA	Validate practicability UML4SOA & usefulness MDD4SOA transformers in practice (by full transformation from UML4SOA to actual code & execution); Transformers now more readable & precise error reporting				Validate approach; Developed hand in hand with UML4SOA
VIATRA2	Test method; Support incremental service development & reusability; XML descriptors created automatically				
GT		Untangle business & presentation logic; New rule set; Architecture migration now automated			
MBTE		Test method; Steered better user guiding			
Patterns	Accompanied by examples from the case study application that identified patterns; Created pattern catalogue documenting (dis)advantages & feedback of approaches				
SDE	Orchestration features employed to combine integrated tools for development & analysis of case studies; Now contains all SENSORIA tools; Now improved SDE interfaces & graphical orchestration mechanism				

5 Relevant SENSORIA Publications

The SENSORIA related publications listed in the References below are not repeated here, since only [BG92] is not related to SENSORIA.

References

[AB08] L. Acciai and M. Boreale. A type system for client progress in a service-oriented calculus. In *Montanari Festschrift*, volume 5065 of *LNCS*, pages 642–658. Springer, 2008.

[Ale09] M. Alessandrini. *Intelligent Service System*. PhD thesis, Westfälische Wilhelms-Universität Münster, 2009.

[BBB09] C. Bodei, L. Brodo, and R. Bruni. Static detection of logic flaws in service-oriented applications. In *ARSPA-WITS*, volume 5511 of *LNCS*, pages 70–87. Springer, 2009.

[BCL+10] M. Bartoletti, L. Caires, I. Lanese, F. Mazzanti, D. Sangiorgi, H.T. Vieira, and R. Zunino. Tools and Verification. In *[WH10]*, LNCS. Springer, 2010.

[BDFZ08] M. Bartoletti, P. Degano, G. Ferrari, and R. Zunino. Model checking usage policies. In *TGC*, volume 5474 of *LNCS*, pages 19–35. Springer, 2008.

[BDFZ09] M. Bartoletti, P. Degano, G. Ferrari, and R. Zunino. Local policies for resource usage analysis. *ACM Transactions on Programming Languages and Systems*, 31(6), August 2009.

[BFG+] L. Bocchi, J.L. Fiadeiro, S. Gilmore, J. Abreu, M. Solanki, and V. Vankayala. A formal approach to modelling time properties of service-oriented systems. Submitted.

[BFL08] L. Bocchi, J.L. Fiadeiro, and A. Lopes. A use-case driven approach to formal service-oriented modelling. In *ISoLA*, volume 17 of *CCIS*, pages 155–169. Springer, 2008.

[BG92] R.M. Burstall and J.A. Goguen. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.

[BLPT09] F. Banti, A. Lapadula, R. Pugliese, and F. Tiezzi. Specification and analysis of SOC systems using COWS: A finance case study. In *WWV*, volume 235 of *ENTCS*, pages 71–105. Elsevier, 2009.

[BM10] M.G. Buscemi and U. Montanari. A prioritised language for QoS negotiation in service composition. Technical report, Dipartimento di Informatica, Università di Pisa, 2010.

[BZ09] M. Bartoletti and R. Zunino. LocUsT: a tool for model checking usage policies. Technical Report TR-08-06, Dipartimento di Informatica, Università di Pisa, 2009.

[CCG+10] I. Cappello, A. Clark, S. Gilmore, D. Latella, M. Loreti, P. Quaglia, and S. Schivo. Quantitative Analysis of Services. In *[WH10]*, LNCS. Springer, 2010.

[CGT08] A. Clark, S. Gilmore, and M. Tribastone. Service-level agreements for service-oriented computing. In *WADT*, volume 5486 of *LNCS*, pages 21–36. Springer, 2008.

[CGT09a] A. Clark, S. Gilmore, and M. Tribastone. Quantitative analysis of web services using SRMC. In *SFM*, volume 5569 of *LNCS*, pages 296–339. Springer, 2009.

[CGT09b] A. Clark, S. Gilmore, and M. Tribastone. Scalable analysis of scalable systems. In *FASE*, volume 5503 of *LNCS*, pages 1–17. Springer, 2009.

[Fos09] H. Foster. Architecture and behaviour analysis for engineering Service Modes. In *PESOS*, pages 1–8. IEEE, 2009.

[GM09a] C. Guidi and F. Montesi. Implementation of the finance case study in Jolie. `http://www.jolie-lang.org/`, 2009.

[GM09b] C. Guidi and F. Montesi. Reasoning about a service-oriented programming paradigm. In *YR-SOC*, volume 2 of *EPTCS*, pages 67–81, 2009.

[GMRMS07] S. Gorton, C. Montangero, S. Reiff-Marganiec, and L. Semini. StPowla: SOA, policies and workflows. In *ICSOC*, volume 4907 of *LNCS*, pages 351–362. Springer, 2007.

[GNN09] H. Gao, F. Nielson, and H.R. Nielson. Protocol stacks for services. In *FCS*, 2009.

[GPT10] S. Gnesi, R. Pugliese, and F. Tiezzi. The SENSORIA pattern-based approach applied to the finance case study. SENSORIA Deliverable Th05, 2010.

[Gua09] R. Guanciale. *The Signal Calculus: Beyond Message-based Coordination for Services*. PhD thesis, IMT Institute for Advanced Studies, Lucca, 2009.

[HKMU06] D. Hirsch, J. Kramer, J. Magee, and S. Uchitel. Modes for software architectures. In *EWSA*, volume 4344 of *LNCS*, pages 113–126. Springer, 2006.

[KMWZ10] A. Knapp, G. Marczyński, M. Wirsing, and A. Zawłocki. A heterogeneous approach to service-oriented systems specification. In *SOAP track at SAC*. ACM, 2010.

[LPT07] A. Lapadula, R. Pugliese, and F. Tiezzi. Regulating data exchange in service oriented applications. In *FSEN*, volume 4767 of *LNCS*, pages 223–239. Springer, 2007.

[MJFT08] P. Mayer, M. Junker, H. Foster, and M. Tribastone. The SDE closeup: Analyzing service-oriented software with the help of formal tools. Technical report, Lehrstuhl PST, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.

[MRMS07] C. Montangero, S. Reiff-Marganiec, and L. Semini. Logic-based detection of conflicts in Appel policies. In *FSEN*, volume 4767 of *LNCS*, pages 257–271. Springer, 2007.

[Str09] D. Strollo. *Designing and Experimenting Coordination Primitives for Service Oriented Computing*. PhD thesis, IMT Institute for Advanced Studies, Lucca, 2009.

[tBBG09] M.H. ter Beek, A. Bucchiarone, and S. Gnesi. Dynamic software architecture development: Towards an automated process. In *SEAA*, pages 105–108. IEEE, 2009.

[tBGMS09] M.H. ter Beek, S. Gnesi, C. Montangero, and L. Semini. Detecting policy conflicts by model checking UML state machines. In *ICFI*, pages 59–74. IOS, 2009.

[tBLLP10] M.H. ter Beek, A. Lapadula, M. Loreti, and C. Palasciano. Analysing Robot Movement using the SENSORIA Methods. In *[WH10]*, LNCS. Springer, 2010.

[tBM10] M.H. ter Beek and F. Mazzanti. Modelling and analysing the finance case study in UMC. Technical report, ISTI-CNR, 2010.

[TG10] M. Tribastone and S. Gilmore. Scaling Performance Analysis using Fluid-Flow Approximation. In *[WH10]*, LNCS. Springer, 2010.

[VCV] H.T. Vieira, L. Caires, and R. Viegas. The Spatial Logic Model Checker. `http://www-ctp.di.fct.unl.pt/SLMC/`.

[WH10] M. Wirsing and M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems—Results of the SENSORIA project on Software Engineering for Service-Oriented Computing*. Springer, 2010.

[XK09] R. Xie and N. Koch. Automotive case study: Demonstrator. Report. Cirquent, 2009.