# SENSORIA Results Applied to the Case Studies*

Maurice H. ter Beek

ISTI–CNR, Pisa, Italy
terbeek@isti.cnr.it

**Abstract.** In this chapter we provide an overview of the application of the results obtained in SENSORIA (i.e., techniques, methods and languages developed in the technical work packages WP1-WP7) to case studies from the Automotive, eUniversity, Finance and Telecommunication domains (developed in work package WP8).

## Introduction

We will not describe the case studies in this chapter, since they are introduced in detail in Chapters 0-3 and 7-1. Likewise we will not describe all details of the various techniques, methods and languages, which can be found in other chapters of this book. The scope of this chapter is rather to emphasize the central role of the case studies in *feeding* and *steering* the research in SENSORIA. This is done by providing a concise overview of the exploitation of SENSORIA results in the case studies. Chapter 7-4 moreover provides an in-depth view of the SENSORIA approach applied to the Finance case study.

This chapter 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 other chapters. As such, the goal of this chapter is to answer the following questions:

**Aim:** besides validating the technique, method or language against 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 what 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?

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

[1] We include applications to the Bowling Robot demonstration described in Chapter 7-3.

M. Wirsing and M. Hölzl (Eds.): SENSORIA Project, LNCS 6582, pp. 655–677, 2011.

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

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

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

The third theme, finally, deals with various engineering aspects of services: *model-driven development* (e.g., MDD4SOA, SDE, VIATRA2), *deployment* (e.g., Modes, Dino, JCaSPiS, Model transformations for deployment) and *reengineering* (e.g., graph transformations). This theme constitutes the work packages WP6 and WP7.

The chapter 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.

## 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 modeling and programming primitives for services.

### 1.1 Architectural Level

The UML2 profile UML4SOA is an implementation- and platform independent means of modeling service-oriented systems, in particular *service interactions* and *orchestrations*. It is described in detail in Chapter 1-1. To test its usefulness in practice, all case studies have been described with UML4SOA, and the resulting diagrams have often been used as the starting point for analyzing the case studies with formal verification frameworks (see Sect. 2). This has led to several improvements, among which shortcuts for SOC patterns and specific support for the soaML profile (created by an OMG task force), which allows the modeling of services, provided and required ports, interfaces, and message types in addition to the behavioral specifications in UML4SOA. Moreover, the need for data handling within UML4SOA diagrams was identified, to 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, which has led to the inclusion of subscoping in UML4SOA as a means to swap out parts of orchestration processes. The profile has been integrated in a model-driven development process described in Sect. 3.

The SENSORIA Reference Modeling Language (SRML) offers primitives for modeling *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 (see Sect. 2). An advantage of SRML that modeling is done at a high level of abstraction, where one can assume that the basic mechanisms of SOAs, like sessions and service discovery/binding, are provided by the middleware and, therefore, need not be part of the models. SRML is described in detail in Chapter 1-2. SRML's primitives and interaction protocols have been validated by modeling scenarios from the Automotive, Finance and Telecommunications case studies [1,2,3,4]. Its application to these case studies also served to validate SRML's three-layered approach (a service layer in between a top and a bottom layer) and the way to define SLAs. The experience with modeling nontrivial interaction protocols has resulted in the addition of key parameters to interactions. The application to the Finance case study served to validate the extension of SRML with timing aspects and the use of PEPA together with SRML to analyze timing properties.

The Service-Targeted Policy-Oriented WorkfLow Approach (StPowla) is a workflow-based approach to *business process modeling* integrating a simple graphical notation, the policy language Appel and the SOA. It is described in detail in Chapter 1-3. To exemplify the approach, StPowla has been applied to a scenario from the Finance case study [5], using UML4SOA to model the workflow. The details of the business process are expressed as compositions of policies written in Appel. Conflicts can be avoided by the analysis techniques of [6,7] (see Sect. 2). An improvement that is the result of specifying this scenario in StPowla is that default templates for the policies are now automatically derived from the workflow, ready to be filled in by the business analyst and converted by the policy server. StPowla has also been applied to the Telecommunications case study [8], in order to assess its impact on the design of business processes.

### 1.2 Programming Level

Among the core calculi developed in SENSORIA and described in Part 2, COWS, SOCK, (Mar)CaSPiS and $\lambda^{req}$ have been applied to the Automotive and Finance case studies and CC has been applied to the Finance case study.

The Calculus for Orchestration of Web Services (COWS) is a modeling notation for all relevant phases of the life cycle of service-oriented applications, among which publication, discovery, SLA negotiation and orchestration. Besides service interactions and compositions, important aspects like *fault* and *compensation handling* can be modeled in COWS. Extensions moreover allow timed activities, constraints and stochastic reasoning. Application to the case studies has demonstrated the feasibility of modeling service-oriented applications with the specific mechanisms and primitives of COWS [9,10,11]. It has moreover triggered the development of an automatic translation from UML4SOA diagrams into COWS and that of COWS-based verification tools (see Sect. 2). The credit request scenario of the Finance case study has also been modeled in sCOWS (see

Chapter 5-5), which is a stochastic extension of COWS [12], allowing quantitative analyses with its related tools (see Sect. 2). 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. Other differences (i.e., the use of service identifiers versus replication, and the adoption of a labeled semantics versus a reduction semantics) are ascribed to meet some technical requirements for the applicability of Markovian techniques.

The Service-Oriented Computing Kernel (SOCK) is a calculus that closely follows recent technologies, with message routing based on *correlation sets* and *request-response service invocations* in WSDL style as primitives. SOCK can therefore easily be implemented on top of common service platforms. This is exploited in the Java Orchestration Language Interpreter Engine Jolie [13], whose properly extended semantics coincide with SOCK. The Automotive case study has been modeled with SOCK and implemented in Jolie [14], to validate SOCK's primitives and, in particular, its extension to model faults and compensations. This was needed to verify in practice its innovative dynamic handler installation and automatic fault notification, which has led to improved handlers [15].

The service-oriented architecture of the credit request scenario from the Finance case study has been fully implemented in Jolie [16], with the aim of showing that Jolie is a mature technology for developing distributed applications by using a fully implemented *service-oriented programming paradigm*. In Jolie everything is a service. Services can be embedded and aggregated to obtain complex services. Hence, designing a service-oriented architecture in Jolie is simple because a programmer is forced to develop only services. The application to the Finance case study has led to the discovery of a new service-oriented architectural pattern, called SoS (Service of Services), in which a service can be considered a proprietary resource of a client rather than of a session. New service-oriented architectural patterns discovered in Jolie are described in detail in [17].

The Calculus of Services with Pipelines and Sessions (CaSPiS) is a calculus for service-oriented applications based on the notions of *sessions* and of *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. CaSPiS is described in detail in Chapter 2-1, which also contains a CaSPiS model of a significant fragment of the credit request scenario from the Finance case study that was used to test the effectiveness of the technique against a concrete and nontrivial example. The credit request scenario and the Bowling Robot case study have also been modeled with MarCaSPiS (see Chapters 5-5 and 7-3), which is a Markovian extension of CaSPiS (see Chapter 2-1), allowing the automatic analysis with its related stochastic logic (see Sect. 2).

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. CC is described in [18] and in Chapter 2-1. Extensions to CC considering exception handling techniques have been studied, showing CC is able to embed *compensable transactions* (see Chapter 3-3). The credit request scenario from the Finance case study has been modeled and typed in CC, thus allowing to verify key properties of the system such as *conversation fidelity*, *progress* and *choreography conformance* (see Sect. 2).

$\lambda^{req}$ is a core functional calculus for services and service orchestration. It is described in detail in Chapter 2-4. $\lambda^{req}$ features primitives for selecting and invoking services that respect given behavioral requirements (*call-by-contract*). Services are modeled 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-called *local policies*. The design methodology has been applied to the on road assistance scenario from the Automotive case study and to the Finance case study [19,20,21], focusing, respectively, on the design of the workflow of the service orchestration and taking into account the specific driver policies and security service contracts and on policies for resource usage.

CaPiTo is a process calculus for modeling 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. It is described in detail in Chapter 4-1. A scenario from the Finance case study has been modeled by CaPiTo [22], showing that CaPiTo is powerful enough to model service-oriented applications, in particular when cryptographic protocols are used to ensure security. It moreover allows the application of the protocol analysis tool LySa to verify the case study (see Sect. 2).

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 [23]. Uncertainty in SOC is understood as different service instances having different performance characteristics due to the inherent heterogeneity of large-scale distributed systems. The expressivity of SRMC was tested by modeling the e-University case study [24] and a novel analysis approach was created to allow precise quantitative statements about such models (see Sect. 2).

The cc-pi calculus is a constraint-based language that supports dynamic selection of services by allowing to specify *QoS negotiations* among service providers and requesters. It combines basic operations of concurrent constraint programming with a symmetric, synchronous mechanism of interaction à la pi-calculus. Chapter 3-1 presents its key features. The cc-pi calculus has been applied to the Telecommunication case study for specifying QoS policies and for enforcing them at execution time [25]. In Chapter 3-1, a variant of the cc-pi calculus that features a choice operator whose branches have a *priority* is proposed, and the credit request scenario from the Finance case study has been modeled in cc-pi, thus allowing for a model of QoS negotiations in which the partners involved have a given order of preference between their possible alternatives.

Architectural Design Rewriting (ADR) is a term-rewriting and graph-based approach to *style-consistent design* and *reconfiguration* of software architectures. It is described in detail in Chapters 1-4 and 3-5. ADR has been applied to scenarios from the Automotive case study [26], to disambiguate the informal specifications of architectural issues by formally modeling architectural constraints and reconfigurations. This has led to an extension of the approach, separating term-rewriting techniques for reconfiguration from those for ordinary behavior, and to specific mechanisms to deal with certain architectural constraints.

A logical specification framework based on *institutions* allows for declarative specifications and modeling of service-oriented architectures. These logics have been applied to the course selection scenario from the eUniversity case study to validate the approach [27]. This has shown that it is useful to separate the description of the behavior of individual services from that of their choreography, which has led to two different logical systems (a local and a global one) formalized as institutions.

## 2 Qualitative and Quantitative Analysis

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

### 2.1 Qualitative Analysis

StPowla's policy language Appel has been applied to the Automotive and Finance case studies. In [6], policies for the on road assistance scenario from the Automotive case study have been defined by *liveness* formulae in a distributed temporal logic and conflicts detected by (semi-) automatic theorem proving. This application has required an extension of the semantics of Appel, which was originally defined only for the non-distributed fragment. In [7], it is shown how to use UMC for the *conflict detection* of policies in a scenario from the Finance case study expressed in UML. This application has required the definition of a correspondence between Appel policies and UML state machines.

CMC and UMC are two prototypical instantiations of a common logical verification framework for the analysis of *functional* properties of service-oriented systems, as described in detail in Chapters 4-2 and 4-3. They only differ with respect to the underlying computational models, which are built from COWS specifications in the case of CMC and from UML statecharts in the case of UMC. The CMC-UMC framework has been used to analyse scenarios from the Automotive, Finance and Telecommunications case studies [9,10,11,28,29,30,31,32] and to the Bowling Robot case study (see Chapter 7-3). This was done by model checking behavioral properties expressed in either the action- and state-based branching-time temporal logic UCTL or its *service-oriented specialization* SocL, for which a set of patterns of service properties was defined. These applications to the case studies have led to a fine-tuning of both the logics and the model-checking approach.

UMC has been integrated in the SRML methodology (see Sect. 1), allowing the model checking of SRML specifications of service compositions based on typical service interaction patterns encoded with UML statecharts [33]. Furthermore, *confidentiality* properties have been verified for the systems modeled in COWS (see Sect. 1) by using its type system, thus showing the feasibility also of type checking. The obtained feedback has 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.

The Verification ENvironment for UML models of Services (VENUS) tool automatically *translates* UML4SOA models of services and natural language statements of service properties into COWS terms and SocL formulae and then checks them using CMC, possibly providing counterexamples. VENUS is described in detail in Chapter 7-4. It has been explicitly developed to *shepherd* the (non-expert) users in writing the behavioral service properties they want to verify. It has been applied to the credit request scenario from the Finance case study in Chapter 7-4.

Both the type system for CaSPiS, developed in [34], and the Control Flow Analysis for CaSPiS, developed in [35], have been applied to the credit request scenario from the Finance case study. The type system aims at statically checking the *client progress* property. This property states that a client-service interaction will not deadlock until the client completes its protocol. The Control Flow Analysis aims instead at detecting and preventing certain *misuses* at the service application logic level. A CaSPiS model of the credit request scenario is reported in Chapter 2-1. Both static techniques and their applications to the scenario are described in Chapter 2-3.

The *static analysis* techniques developed for CC, described in [18] and in Chapter 2-3, single out systems where multiple parties interacting in a conversation follow the prescribed protocols of interaction, even when some of them have dynamically joined the conversation, and systems that are free from deadlocks, even when participants are simultaneously involved in several (even dynamically acquired) conversations. Using the credit request scenario from the Finance case study modeled and typed in CC (see Sect. 1), it has been proven that the multiparty interaction in the credit request system follows *well-defined* protocols of interaction and is *free from deadlocks* (see Chapters 2-1 and 2-3).

The chorSLMC tool has been developed to support the verification of *choreography conformance* by translating CC specifications in a dialect of the $\pi$-Calculus and WS-CDL-like choreography descriptions in dynamic spatial logic formulae, which then allows the use of the Spatial Logic Model Checker SLMC [36] to check a system's conformance to the given choreography, among other properties. This is described in detail in Chapter 4-3. The credit request system from the Finance case study has been checked to conform to the prescribed choreography using the chorSLMC tool, based on the CC implementation presented in Chapter 2-1 and on the Conversation Types shown in Chapter 2-3 (see Sect. 1).

The protocol analysis tool LySa [37] has been applied to the CaPiTo model of a scenario from the Finance case study [22] (see Sect. 1 and Chapter 4-1). This analysis suggests that the *authentication* property holds: Once the decision has been made as to whether the request has to be validated by the service, it cannot be tricked into being processed by the wrong one. The analysis moreover suggests that no sensitive data is leaked to the attacker, hence *confidentiality* holds as well. A number of further static analyses have been applied to process-algebraic models of the Automotive case study [38,39,40], with the aim of validating *privacy-related* properties, correct *service delivery* and proper *message correlation*, respectively.

Event-based Service Coordination (ESC) is a middleware supporting the design and implementation of service coordination policies (both *orchestration* and *choreography*). 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. The SC-ESC framework provides a variety of techniques that are mathematically rigorous and pragmatically useful, and which enable the implementation of SOC systems. The framework is described in detail in Chapter 3-4. Scenarios from the Automotive and Finance case studies have been specified and implemented in the SC-ESC framework [41,42,43,44,45]. The management of the case studies has allowed to experiment, evaluate and reason about long-running transactions.

Open Consume-Produce-Read (OCPR) nets are a type of Petri nets that can model the behavior of OWL-S web services through a direct mapping [46]. A compositional notion of equivalence between web services represented as OCPR nets has been applied to several scenarios from the Automotive and Finance case studies [46,47]. It has been employed for checking whether or not a service specification is *equivalent* to a service implementation, and whether or not one (sub)service may *replace* another (sub)service without altering the behavior of the whole application. The need to address the asymmetry of the matching of services (i.e., to check whether a (composition of) service(s) matches a query that specifies the behavior of the desired service (composition) to be found) triggered the introduction of simulation, thus taking into account the chance of satisfying a query with an overspecified service.

The LTSA WS-Engineer tool provides support for a model-based approach to verifying compositions of service architectures, behavior and deployment configurations. The tool supports verification of properties created from design specifications and implementation models to analyze *correctness* and *consistency* of *service compositions*. LTSA WS-Engineer supports verification of service compositions with design (in the form of MSCs), interactions (between multiple services), choreography (in the form of WS-CDL) and deployment models (in the form of xADL2 or UML2). It has been applied to scenarios from the Automotive, Finance and eUniversity case studies [48,49,50] as well as to the Bowling Robot case study (see Chapter 7-3), which has helped greatly to further develop the tool.

### 2.2 Quantitative Analysis

The Mobile Stochastic Logic (MoSL) has been applied to the accident assistance scenario from the Automotive case study [51,52], with the aim of validating the expression of non-functional, *performance* and *dependability-oriented* properties/requirements of/on services. In particular, a complex *responsiveness* property is developed in detail. The major benefit of using MoSL is the possibility of using arbitrarily nested logical operators: This allows the expression of properties whose formulation would be very difficult and error-prone in natural language. Furthermore, MoSL allows both functional and non-functional requirements to be expressed in an integrated way.

The Service-oriented Stochastic Logic (SoSL), a variant of MoSL, was designed for dealing with specific SOC features. It has been applied to the MarCaSPiS model of the credit request scenario from the Finance case study (see Sect. 1), as described in Chapter 5-1. Three specific aspects have been addressed: system *performance*, supervisor and employee *workload*, and system *reactivity*. The probability for clients to get served as a function of (waiting) time has been studied and such a probability dramatically decreases when the number of clients increases. Supervisor and employee workload has been found rather low, showing that the service is under-used. On the down side, only scenarios with few clients active at the same time could be analyzed due to the size of the state space, a problem that could be tackled with the use of MarCaSPiS discrete simulation or Ordinary Differential Equations semantics. SoSL has also been applied to the MarCaSPiS model of the Bowling Robot case study (see Sect. 1), as described in Chapter 7-3. This latter analysis has focused on the probability of the set of computations that lead to a state satisfying a generic property, showing that the methodology using MarCaSPiS provides both "intuitive" and effective estimations of the robot's behavior (with respect to variations of the model's parameters) as well as a formal basis for reasoning about functional properties of the robot.

$\lambda^{req}$ is equipped with a formal methodology that makes secure orchestration feasible. An abstraction of the program behavior is first extracted, through a type and effect system. This abstract behavior over-approximates the possible runtime histories of all services involved in an orchestration. The abstract behavior is then model checked to construct the skeleton structure of the orchestrator that securely coordinates the running services. Starting from $\lambda^{req}$ models of scenarios from the Automotive and Finance case studies (see Sect. 1), it has been shown that the awareness of security from the early stages of development will foster security through all the following phases of software production. In particular, the output of the model checker has been exploited to highlight design flaws, suggesting how to revise the orchestration and the security policies.

Software tools for the well-known stochastic process algebra PEPA have been applied to almost all case studies: *safety*, *response-time* and *passage-end* analyses have been performed on scenarios from the Automotive case study [53,54,55], which has led to a more user-friendly PEPA development environment and to integration with the SDE (see Sect. 3). Passage-end analysis was also applied to

the Bowling Robot case study (see Chapters 5-4 and 7-3). Moreover, *response-time* and *sensitivity* analyses have been applied to the credit request scenario from the Finance case study (see Chapter 5-5), with the intention of identifying the bottleneck activity: 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. Such investigations allow one to decide where effort would be best spent in improving functions within the entire business process. 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). Finally, the eUniversity case study has been used to illustrate the main theoretical developments of PEPA with respect to its deterministic interpretation. One of these is described in detail in Chapter 5-3: A modeling strategy which may be applied to a variety of distributed applications, including service-oriented ones. This has revealed as one of PEPA's key benefits, its capability of targeting both a stochastic and a deterministic semantics, and it has prompted further work in the area of software tool development.

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 modeling and analysis tools [56]. A number of qualitative analyses have been performed on the SRMC model of the eUniversity case study (see Sect. 1), among which the scalability of the eUniversity system in the presence of increasing numbers of student users and uncertainty about system configuration [24].

Two distinct tools have been developed to assist the quantitative analyses of sCOWS specifications. One of them, called sCOWS_LTS, allows sCOWS probabilistic model checking through the generation of the LTS (Labeled 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. Both tools have been used to analyze the system performance of the COWS model of the Finance case study's credit request scenario (see Chapter 5-5).

## 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, methods for reengineering legacy systems into service-oriented ones, and a software development process for service-oriented systems.

### 3.1 Deployment and Reengineering

The notion of software architecture Modes in the context of SOC abstracts a set of services, and their states, collaborating towards a common goal. A Mode can be used to identify the services required in a service composition, and assist in specifying *orchestration* and *choreography* requirements through service component state changes. Modes are described in detail in Chapter 4-4. The concept of Modes complements that of ADR (see Sect. 1): A Mode is an architectural style whose productions focus on architectural change and behavioral correctness. Modes have been applied to the route planning scenario from the Automotive case study [57,50], which has led to the development of a UML2 Modes profile. Moreover, the approach has been extended to Service Mode analysis (see Chapter 4-4), including extensions to the WS-Engineer tool (see Sect. 2).

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. Dino is described in detail in Chapter 6-3. Dino has been applied to the several scenarios from the Automotive case study [58], in order to provide the requirements, guide the design, and help validate the work done in the Dino project (see Chapter 6-3). This has allowed the design to be validated and improved. 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 [59].

JCaSPiS is a Java framework that permits implementing service-oriented applications based on the CaSPiS paradigm [60]. Indeed, JCaSPiS provides a set of classes that implements primitives for publishing and invoking services, for defining protocols used to control the service interactions, and mechanisms for the handling of unexpected behaviors (session closures). Starting from the CaSPiS specification of the credit request scenario from the Finance case study (see Sect. 1), a *real-world* scenario was implemented with JCaSPiS. This allows programmers to use a formal language to model system components and their interactions and, moreover, analyses performed at the level of specification are preserved at the level of implementation. The application to the case study has led to the development of new functionalities.

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). It is described in detail in Chapter 1-1 (see also the section on model-driven development below). The model transformers have been applied to scenarios from all case studies [61,62,63], with the exception of the Bowling Robot case study. The aim was to validate the practicability of the UML4SOA profile and, in particular, the usefulness of the MDD4SOA transformers in practice (by means of a *complete transformation* from an UML4SOA model down to actual code and its execution on a target platform). Obviously, the development of MDD4SOA has gone hand in hand with that of UML4SOA. The transformers have shown their use in SENSORIA for converting UML models to input languages for analysis and for using the

converted models as the basis for the implementation of the case studies. These uses have themselves led to improvements of the transformers, among which more readable and precise error reporting.

A general methodology for *architectural migration* based on Graph Transformations (GT) takes a graph model of the source code, after categorizing blocks of source code according to the target architecture, and transforms these into a graph model of the target architecture, through the creation of a metamodel-based representation of the code, and, finally, into the target code. It is described in detail in Chapter 6-4. The methodology has been applied to a scenario from the Finance case study [64]. This scenario has a two-tier architecture (client/server) but presentation, business logic and data access code elements are all mixed in both tiers. By applying the GT rules it was possible to *untangle* these concerns and obtain a well-organized model, following the service-orientation requirement of separation between business and presentation logic. This has led to an improvement of the existing GT rule set and to new GT rules, thus increasing the number of supported situations. The resulting rule set can be used to automate part of the process in projects that involve architecture migration.

The VIATRA2 *model transformation* framework forms the basis of a model-driven method for facilitating the development and deployment of services with security and reliable communication requirements. It is described in detail in Chapters 6-1 and 6-2. *Deployment transformations*, extended with some Java-based template generation, take UML4SOA models as input and produce standards-compliant service descriptors (WSDL files) and configuration descriptors to reliable and secure service middleware of the Apache platform. This method has been tested on scenarios from the Automotive, Finance and eUniversity case studies. This has shown that while the size of these models does not necessarily require a sophisticated model-driven approach, incremental development of services and reusability of components is strongly supported by this method, which is a clear benefit for such projects. Moreover, the time consuming and error-prone task of creating XML descriptors is done automatically.

### 3.2 Model-Driven Development

The model-driven engineering approach MDD4SOA (described above) 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). The transformations have been applied to the thesis management scenario from the eUniversity case study and to the on road assistance scenario from the Automotive case study for testing purposes [62,63]. In particular, the Automotive Demonstrator focuses on the development process of service-oriented software [63], demonstrating how

a model-driven process can work. The feedback has led to improvements of the UML4SOA metamodel and profile as well as of MDD4SOA.

Model Transformation By Example (MTBE) is a model-driven software engineering approach to *derive* model transformation rules from an initial prototypical set of interrelated source and target models, which describe critical cases of the transformation in a purely declarative way [65,66]. Its applicability was assessed by trying to reproduce the model transformation rules for the UML2SOA transformation as part of VIATRA (see above). As sample input model the UML component model of the credit request scenario from the Finance case study was used. The sample output model was generated by the original VIATRA transformation (previously developed by hand). Using MTBE, it was possible to derive most transformation rules in the form of graph transformation rules in the VIATRA framework directly from the UML models of the scenario. This has steered research for better guiding the users when an MTBE run does not provide meaningful results so that the training models can be changed appropriately. The VIATRA2 transformation framework was also extended with *incremental pattern matching mechanisms*, helping fast execution of model transformations.

The BPEL2SAL *transformation chain*, which has been implemented in VIATRA2 (see above), facilitates model-driven analysis of service orchestrations. Symbolic Analysis Laboratory (SAL) is a verification framework that is directed at analyzing properties of transition systems by combining tools for program analysis, model checking, and theorem proving [67]. BPEL2SAL is described in detail in Chapter 6-1. As a distinctive feature, the transformations are extended with enhanced traceability techniques which facilitate the visualization of model-checking results directly on the Eclipse BPEL designer tool. The method has been applied on scenarios from the Finance and eUniversity case studies.

The SENSORIA Pattern language describes which problems are addressed by SENSORIA techniques and tools, how they *solve* the *problems* they address, and which *forces* determine whether or not a technique or tool is appropriate for a given *situation*. It moreover does so in a manner accessible to software developers not involved in SENSORIA. The language is described in detail in Chapter 7-5. Each pattern is accompanied by examples taken from the case study application in which the pattern was identified [68], and which have been used to validate the pattern. The pattern catalogue serves to document the advantages and disadvantages of different approaches and to enhance the discussion between SENSORIA partners. Finally, since the patterns are derived from the case studies development, they are strongly influenced by feedback from these case studies.

The SENSORIA Development Environment (SDE) is a *tool integration platform* for the 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. It is described in detail in Chapter 6-5. The orchestration features within the SDE have been employed for combining several integrated tools for the development and analysis of

the case studies. 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 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. In [63], 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 [49].

## 4 Concluding Overview

In Table 1, we provide a synthetic overview of the case studies to which the SENSORIA techniques, methods and languages have been applied. 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 on in the project. Second, the Bowling Robot *demonstration* has been introduced rather late in 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 we provide more detailed analytic overviews of the specific experience/benefits of having applied the SENSORIA techniques, methods and languages to the case studies, organized in the format of one table per theme. These tables thus contain very brief summaries of the answers to the four questions (aim, experience, benefits, feedback) that we presented in the Introduction as the goal of this chapter. We have compiled these summaries based on the answers provided by the partners that have developed and used the various SENSORIA techniques, methods and languages.

**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 | | √ | | | |
| | $\lambda^{req}$ | √ | √ | | | |
| E | CaPiTo | | √ | | | |
| | SRMC | | | √ | | |
| 1 | cc-pi | | √ | | | √ |
| | ADR | √ | | | | |
| | Institutions | | | √ | | |
| T | SocL | √ | √ | | | |
| H | CMC-UMC | √ | √ | | √ | √ |
| | VENUS | | √ | | | |
| E | chorSLMC | | √ | | | |
| | LySa | | √ | | | |
| M | SC-ESC | √ | √ | | | |
| | OCPR | √ | √ | | | |
| E | WS-Engineer | √ | √ | √ | √ | |
| | SoSL/MoSL | √ | √ | | √ | |
| 2 | PEPA toolkit | √ | √ | √ | √ | |
| | Modes | √ | | | | |
| T | Dino | √ | √ | | | |
| H | JCaSPiS | | √ | | | |
| E | MDD4SOA | √ | √ | √ | | √ |
| M | GT | | √ | | | |
| E | VIATRA2 | √ | √ | √ | | |
| | MTBE | | √ | | | |
| 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 verification; Now shortcuts SOC patterns, support soaML profile, data handling & subscoping | | | | |
| 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<br>sCOWS | Feasibility mechanism + primitives to model SOA & provide stochastic description | | | | |
| SOCK | Validate primitives & faults + compensations modeling; Verify dynamic handler & automatic fault notification; Now improved handlers | | | | |
| Jolie | Test programming a SOA; Found new SOA patterns | | | | |
| CaSPiS<br>MarCaSPiS | Test its effectiveness & its Markovian extension | | | Feasibility of methodology | |
| CC | | Type to verify key properties | | | |
| $\lambda^{req}$ | 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 behavioral 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 (VENUS) & UMC | | | Test & fine-tune model checker; Verify properties; Now automatic translations from UML4SOA to UMC | |
| VENUS | | Assist user to write properties to verify | | | |
| 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 | | | | |
| LySa | | 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*)

| 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 behavior | |
| $\lambda^{req}$ | 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 PEPA wrt deterministic semantics; Key benefit PEPA: target both 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<br>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 modeling 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 modeling 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 |
| GT | | Untangle business & presentation logic; New rule set; Architecture migration now automated | | | |
| VIATRA2 | Test method; Support incremental service development & reusability; XML descriptors created automatically; Now BPEL2-SAL orchestration analysis back-annotated | | | | |
| MTBE | | 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 | | | | |

**Acknowledgments.** We thank our partners in the SENSORIA project for their contributions to this chapter.

## References

1. Abreu, J., Bocchi, L., Fiadeiro, J.L., Lopes, A.: Specifying and composing interaction protocols for service-oriented system modelling. In: Derrick, J., Vain, J. (eds.) FORTE 2007. LNCS, vol. 4574, pp. 358–373. Springer, Heidelberg (2007)
2. Bocchi, L., Fiadeiro, J.L., Lopes, A.: Service-oriented modelling of automotive systems. In: COMPSAC, pp. 1059–1064. IEEE, Los Alamitos (2008)
3. Bocchi, L., Fiadeiro, J.L., Lopes, A.: A use-case driven approach to formal service-oriented modelling. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 155–169. Springer, Heidelberg (2008)
4. Bocchi, L., Fiadeiro, J.L., Gilmore, S., Abreu, J., Solanki, M., Vankayala, V.: A formal approach to modelling time properties of service-oriented systems (submitted)
5. Gorton, S., Montangero, C., Reiff-Marganiec, S., Semini, L.: StPowla: SOA, policies and workflows. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 351–362. Springer, Heidelberg (2009)
6. Montangero, C., Reiff-Marganiec, S., Semini, L.: Logic-based detection of conflicts in Appel policies. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 257–271. Springer, Heidelberg (2007)
7. ter Beek, M.H., Gnesi, S., Montangero, C., Semini, L.: Detecting policy conflicts by model checking UML state machines. In: ICFI 2009, pp. 59–74. IOS (2009)
8. Fantini, P., Montangero, C., Palasciano, C., Reiff-Marganiec, S., Semini, L.: Supporting user-friendly design of flexible business processes in StPowla. Technical Report PISATR0825, Dipartimento di Informatica, Università di Pisa (2008)
9. Fantechi, A., Gnesi, S., Lapadula, A., Mazzanti, F., Pugliese, R., Tiezzi, F.: A model checking approach for verifying COWS specifications. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 230–245. Springer, Heidelberg (2008)
10. Lapadula, A., Pugliese, R., Tiezzi, F.: Specifying and analysing SOC applications with COWS. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 701–720. Springer, Heidelberg (2008)
11. Gnesi, S., Pugliese, R., Tiezzi, F.: The SENSORIA pattern-based approach applied to the finance case study. SENSORIA Deliverable Th05 (2010)
12. Prandi, D., Quaglia, P.: Stochastic COWS. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 245–256. Springer, Heidelberg (2007)
13. Montesi, F., Guidi, C., Zavattaro, G.: Composing services with JOLIE. In: ECOWS 2007, pp. 13–22. IEEE, Los Alamitos (2007)
14. Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: On the interplay between fault handling and request-response service invocations. In: ACSD 2008, pp. 190–199. IEEE, Los Alamitos (2008)
15. Montesi, F., Guidi, C., Lanese, I., Zavattaro, G.: Dynamic fault handling mechanisms for service-oriented applications. In: ECOWS 2008, pp. 225–234. IEEE, Los Alamitos (2008)
16. Guidi, C., Montesi, F.: Implementation of the finance case study in Jolie (2009), `http://www.jolie-lang.org/`

17. Guidi, C., Montesi, F.: Reasoning about a service-oriented programming paradigm. In: ter Beek, M.H. (ed.) YR-SOC 2009. EPTCS, vol. 2, pp. 67–81 (2009)
18. Vieira, H.T.: A Calculus for Modeling and Analyzing Conversations in Service-Oriented Computing. PhD thesis, Universidade Nova de Lisboa (2010)
19. Bartoletti, M., Degano, P., Ferrari, G., Zunino, R.: Secure service orchestration. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 4677, pp. 24–74. Springer, Heidelberg (2007)
20. Bartoletti, M., Degano, P., Ferrari, G., Zunino, R.: Semantics-based design for secure web services. IEEE Transactions on Software Engineering 34, 33–49 (2008)
21. Bartoletti, M., Degano, P., Ferrari, G., Zunino, R.: Local policies for resource usage analysis. ACM Transactions on Programming Languages and Systems 31 (2009)
22. Gao, H., Nielson, F., Nielson, H.R.: Protocol stacks for services. In: FCS (2009)
23. Clark, A., Gilmore, S., Tribastone, M.: Service-level agreements for service-oriented computing. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 21–36. Springer, Heidelberg (2009)
24. Clark, A., Gilmore, S., Tribastone, M.: Scalable analysis of scalable systems. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 1–17. Springer, Heidelberg (2009)
25. Buscemi, M.G., Ferrari, L., Moiso, C., Montanari, U.: Constraint-based policy negotiation and enforcement for telco services. In: TASE 2007, pp. 463–472. IEEE, Los Alamitos (2007)
26. Bruni, R., Lluch Lafuente, A., Montanari, U., Tuosto, E.: Style-based architectural reconfigurations. Bulletin of the EATCS 94, 161–180 (2008)
27. Knapp, A., Marczyński, G., Wirsing, M., Zawłocki, A.: A heterogeneous approach to service-oriented systems specification. In: SOAP track at SAC 2010. ACM, New York (2010)
28. ter Beek, M.H., Gnesi, S., Mazzanti, F., Moiso, C.: Formal modelling and verification of an asynchronous extension of SOAP. In: ECOWS 2006, pp. 287–296. IEEE, Los Alamitos (2006)
29. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 133–148. Springer, Heidelberg (2008)
30. ter Beek, M.H., Gnesi, S., Koch, N., Mazzanti, F.: Formal verification of an automotive scenario in service-oriented computing. In: ICSE 2008, pp. 613–622. ACM, New York (2008)
31. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Dynamic software architecture development: Towards an automated process. In: SEAA 2009, pp. 105–108. IEEE, Los Alamitos (2009)
32. ter Beek, M.H., Mazzanti, F.: Modelling and analysing the finance case study in UMC. Technical Report 2010-TR-007, ISTI-CNR (2010)
33. Abreu, J., Mazzanti, F., Fiadeiro, J.L., Gnesi, S.: A model-checking approach for service component architectures. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS 2009. LNCS, vol. 5522, pp. 219–224. Springer, Heidelberg (2009)
34. Acciai, L., Boreale, M.: A type system for client progress in a service-oriented calculus. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 642–658. Springer, Heidelberg (2008)
35. Bodei, C., Brodo, L., Bruni, R.: Static detection of logic flaws in service-oriented applications. In: Degano, P., Viganò, L. (eds.) ARSPA-WITS 2009. LNCS, vol. 5511, pp. 70–87. Springer, Heidelberg (2009)

36. Vieira, H.T., Caires, L., Viegas, R.: The Spatial Logic Model Checker v2.01 (November 2009), http://www-ctp.di.fct.unl.pt/SLMC/
37. Buchholtz, M., Nielson, H.R.: LySa tool v2.02 (October 2006), http://www.imm.dtu.dk/English/Research/Language-Based_Technology/Software/LySaTool.aspx
38. Nielson, H.R., Nielson, F.: A flow-sensitive analysis of privacy properties. In: CSF 2007, pp. 249–264. IEEE, Los Alamitos (2007)
39. Nielson, F., Nielson, H.R., Bauer, J., Nielsen, C.R., Pilegaard, H.: Relational analysis for delivery of services. In: Barthe, G., Fournet, C. (eds.) TGC 2007. LNCS, vol. 4912, pp. 73–89. Springer, Heidelberg (2008)
40. Bauer, J., Nielson, F., Nielson, H.R., Pilegaard, H.: Relational analysis of correlation. In: Alpuente, M., Vidal, G. (eds.) SAS 2008. LNCS, vol. 5079, pp. 32–46. Springer, Heidelberg (2008)
41. Ciancia, V., Ferrari, G., Guanciale, R., Strollo, D.: Checking correctness of transactional behaviors. In: Suzuki, K., Higashino, T., Yasumoto, K., El-Fakih, K. (eds.) FORTE 2008. LNCS, vol. 5048, pp. 134–148. Springer, Heidelberg (2008)
42. Ferrari, G., Guanciale, R., Strollo, D., Tuosto, E.: Event-based service coordination. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 312–329. Springer, Heidelberg (2008)
43. Ferrari, G., Guanciale, R., Strollo, D., Tuosto, E.: Refactoring long running transactions. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 127–142. Springer, Heidelberg (2009)
44. Strollo, D.: Designing and Experimenting Coordination Primitives for Service Oriented Computing. PhD thesis, IMT Institute for Advanced Studies, Lucca (2009)
45. Guanciale, R.: The Signal Calculus: Beyond Message-based Coordination for Services. PhD thesis, IMT Institute for Advanced Studies, Lucca (2009)
46. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: On the use of behavioural equivalences for web services' development. Fundamenta Informaticae 89, 479–510 (2008)
47. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A net-based approach to web services publication and replaceability. Fundamental Informaticae 94, 305–330 (2009)
48. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Tool support for model-based engineering of web service compositions. In: ICWS 2005, pp. 95–102. IEEE, Los Alamitos (2005)
49. Mayer, P., Junker, M., Foster, H., Tribastone, M.: 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)
50. Foster, H.: Architecture and behaviour analysis for engineering Service Modes. In: PESOS 2009, pp. 1–8. IEEE, Los Alamitos (2009)
51. De Nicola, R., Katoen, J.P., Latella, D., Loreti, M., Massink, M.: Model checking mobile stochastic logic. Theoretical Computer Science 382, 42–70 (2007)
52. De Nicola, R., Katoen, J.P., Latella, D., Loreti, M., Massink, M.: Stochastic logics. SENSORIA Deliverable 4.2a (February 2007)
53. Clark, A., Gilmore, S.: Evaluating quality of service for service level agreements. In: Brim, L., Haverkort, B.R., Leucker, M., van de Pol, J. (eds.) FMICS 2006 and PDMC 2006. LNCS, vol. 4346, pp. 181–194. Springer, Heidelberg (2007)
54. Argent-Katwala, A., Clark, A., Foster, H., Gilmore, S., Mayer, P., Tribastone, M.: Safety and response-time analysis of an automotive accident assistance service. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 191–205. Springer, Heidelberg (2008)
55. Clark, A., Duguid, A., Gilmore, S.: Passage-end analysis. In: Bradley, J.T. (ed.) EPEW 2009. LNCS, vol. 5652, pp. 110–115. Springer, Heidelberg (2009)

56. Clark, A., Gilmore, S., Tribastone, M.: Quantitative analysis of web services using SRMC. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 296–339. Springer, Heidelberg (2009)
57. Hirsch, D., Kramer, J., Magee, J., Uchitel, S.: Modes for software architectures. In: Gruhn, V., Oquendo, F. (eds.) EWSA 2006. LNCS, vol. 4344, pp. 113–126. Springer, Heidelberg (2006)
58. Mukhija, A., Dingwall-Smith, A., Rosenblum, D.S.: QoS-aware service composition in Dino. In: ECOWS 2007, pp. 3–12. IEEE, Los Alamitos (2007)
59. Alessandrini, M.: Intelligent Service System. PhD thesis, Westfälische Wilhelms-Universität Münster (2009)
60. Bettini, L., De Nicola, R., Loreti, M.: Implementing session centered calculi. In: Wang, A.H., Tennenholtz, M. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 17–32. Springer, Heidelberg (2008)
61. Foster, H., Mayer, P.: Leveraging integrated tools for model-based analysis of service compositions. In: ICIW 2008, pp. 72–77. IEEE, Los Alamitos (2008)
62. Mayer, P., Schroeder, A., Koch, N.: MDD4SOA: Model-driven service orchestration. In: EDOC 2008, pp. 203–212. IEEE, Los Alamitos (2008)
63. Xie, R., Koch, N.: Automotive case study: Demonstrator. Report. Cirquent (2009)
64. Heckel, R., Correia, R., Matos, C.M.P., El-Ramly, M., Koutsoukos, G., Andrade, L.F.: Architectural transformations: From legacy to three-tier and services. In: Software Evolution, pp. 139–170. Springer, Heidelberg (2008)
65. Varró, D.: Model transformation by example. In: Wang, J., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 410–424. Springer, Heidelberg (2006)
66. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. Software and System Modeling 8, 347–364 (2009)
67. Shankar, N.: Symbolic analysis of transition systems. In: Gurevich, Y., Kutter, P.W., Vetta, A., Thiele, L. (eds.) ASM 2000. LNCS, vol. 1912, pp. 287–302. Springer, Heidelberg (2000)
68. Wirsing, M., Hölzl, M.M., Acciai, L., Banti, F., Clark, A., Fantechi, A., Gilmore, S., Gnesi, S., Gönczy, L., Koch, N., Lapadula, A., Mayer, P., Mazzanti, F., Pugliese, R., Schroeder, A., Tiezzi, F., Tribastone, M., Varró, D.: SENSORIA patterns: Augmenting service engineering with formal analysis, transformation and dynamicity. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 170–190. Springer, Heidelberg (2008)