Sensoria – Software Engineering for Service-Oriented Overlay Computers

Martin Wirsing, Matthias Hölzl, Nora Koch, Philip Mayer

Ludwig-Maximilians-Universität München, Germany
{wirsing,hoelzl,koch,mayer}@pst.ifi.lmu.de

Abstract. Service-Oriented Computing is a paradigm where services are understood as autonomous, platform-independent computational entities that can be described, published, categorised, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems and applications. These characteristics have pushed service-oriented computing towards nowadays widespread success, demonstrated by the fact that many large companies invested a lot of efforts and resources to promote service delivery on a variety of computing platforms, mostly through the Internet in the form of Web services. In the past, service-oriented computing and development has been done in a pragmatic, mostly ad-hoc way. Theoretical foundations were missing that are needed for trusted interoperability, predictable compositionality, and quality issues like security, correctness, or resource usage. The IST-FET integrated project Sensoria has addressed these issues by developing a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach, supporting semi-automatic development and deployment of self-adaptable (composite) services.

1 Introduction

Selling services rather than hardware or software has become the biggest growth business in the computing industry. Business in this area has already evolved from relatively simple (customer) services to global complex (business) solutions. Computing is becoming a utility and software a service. This trend is changing the economics of IT industry and influences the e-Society as a whole.

In the service-oriented computing (SOC) paradigm, services are understood as autonomous, platform independent computational entities that can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems. Today, services are being delivered on a variety of computing platforms, mostly through the Web, Personal Digital Assistants, and mobile phones. Tomorrow, they will be delivered on all kinds of global computers and a plethora of new services will be required for e-government, e-health, and e-science, just to name a few of the areas that are already taking shape within the Information Society. Thanks to their ability to be dynamically assembled, services can provide a much required layer of integration between the different global computers that are being studied and proposed by industry and academia for supporting the operation of the future Information Society. As a result, service-oriented computing is bound to play the role of an ideal overlay computer for Global Computing.

In the past, service-oriented computing and development has been done in a mostly ad-hoc way. Theoretical foundations for trusted interoperability, predictable compositionality, and quality issues like security, correctness, or resource usage were not well-established and service-oriented software development was not integrated in a controllable process based on powerful analysis and verification tools. Furthermore, it was not clear whether formal approaches to service-oriented software development would scale up to the development of large, complex systems.

In order to answer these questions, Sensoria has developed a novel comprehensive approach to the engineering of software systems for service-oriented overlay computers where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. This approach is focused on global services that are context-adaptive, personalisable, and may require hard and soft constraints on resources and performance; it takes into account the fact that services have to be deployed on different, possibly interoperating global computers to provide novel and reusable service-oriented overlay computers.

The results of Sensoria include a new generalised concept of service for global overlay computers, new semantically well-defined modelling and programming primitives for services, new powerful mathematical analysis and verification techniques, tools for system behaviour and quality of service properties, and novel model-based transformation and development techniques. The innovative methods of Sensoria are demonstrated by application in the service-intensive areas of e-business, automotive systems, and e-university.

This chapter is structured as follows: In section **??**, we introduce the Sensoria project with its aims and contributions to the field of service-oriented computing. Sections **??** to **??** outline the three main research themes of Sensoria and provide pointers to the remaining parts of the book. We conclude in section **??**.

2 Sensoria – Well-Founded SOC Development

The core aim of the Sensoria EU project was the production of new knowledge for systematic and scientifically well-founded methods of service-oriented software development. Sensoria provides a comprehensive approach to design, formal analysis, automated deployment, and reengineering of service-oriented applications. The research themes of Sensoria therefore range across the whole lifecycle of software development. Sensoria methods and tools rely on mathematical theories and methods that allow rigorous verification of SOA artifacts. Realistic case studies for different important application areas including telecommunications, automotive, e-learning, and e-business are defined by the industrial partners have been used to verify the applicability of Sensoria methods to industrial domains.

In Sensoria a model-driven approach to service development was chosen, as it enables developers to control the variety of specific distributed, global computing platforms, and to ensure a strict separation of concerns that can break the complexity of service composition and evolution. In this approach, services are first modelled in a platform-independent architectural design layer; these models are then analysed using formal methods and refined; afterwards, they can be used for generating implementations over different global computing platforms in a (semi-)automated way. This is shown in Fig. **??**.

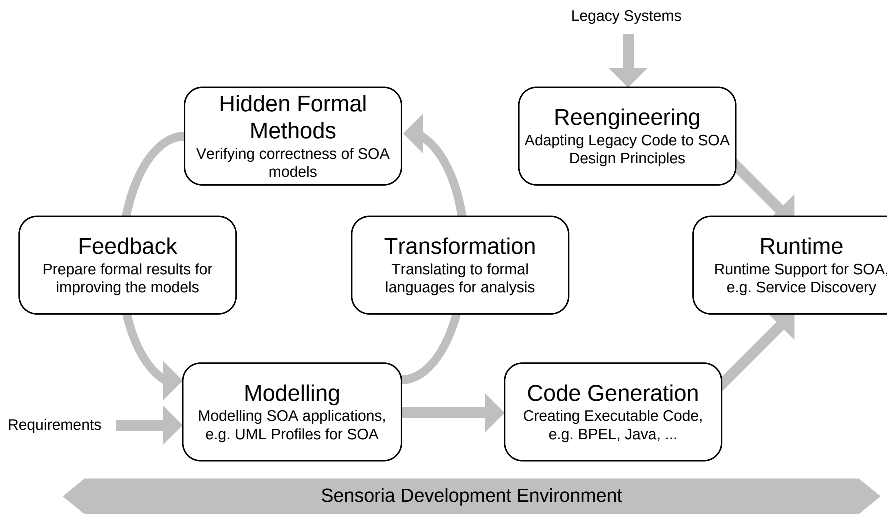
Legacy Systems
Hidden Formal Methods
Verifying correctness of SOA models
Reengineering
Adapting Legacy Code to SOA Design Principles
Feedback
Prepare formal results for improving the models
Transformation
Translating to formal languages for analysis
Runtime
Runtime Support for SOA, e.g. Service Discovery
Modelling
Modelling SOA applications, e.g. UML Profiles for SOA
Code Generation
Creating Executable Code, e.g. BPEL, Java, ...
Requirements
Sensoria Development Environment

Fig. 1. Sensoria support for the model-driven development process

In more detail, the main technical ingredients that Sensoria provides to the service engineer are:

Modelling. Service-oriented applications are designed using high-level visual formalisms such as the industry standard UML or domain-specific modelling languages to precisely capture domain-specific requirements.

Transformation and Feedback. Formal representations are generated by automated model transformations from engineering models.

Hidden Formal Methods. Back-end mathematical model analysis is used to reveal performance bottlenecks, or interactions leading to errors or violation of service contracts. For critical services, the developers can perform deep semantic analysis for certification.

Feedback. Feedback from the formal analysis is presented to the developer in a way that is easy to understand and used to improve the engineering models.

Code Generation. The high-level models are used to generate executable code, e.g., in Java or BPEL, and configuration data for deploying the resulting services to various standards-compliant service platforms.

Runtime. The generated code can utilize advanced run-time infrastructure that was developed as part of the SENSORIA project, e.g., dynamic service brokering, as well as standard service platforms.

Reengineering of legacy services. Many existing systems are built as monolithic non-extensible applications which cannot be easily adapted to new business processes. SENSORIA develops methods to transform these applications into layered systems with well-defined service interfaces.

SENSORIA *Development Environment.* The SENSORIA Development Environment supports the above activities by providing an Eclipse-based, fully-customizable tool chain for the entire model-driven workflow.

These individual development activities have been grouped into three major themes that served as drivers for the scientific research of the SENSORIA project; they also provide the structure for the remainder of this book:

Linguistic Primitives for Modelling and Programming SOA systems. Language primitives for services and their interactions have been developed on two different abstraction levels, an architectural design level and a programming abstraction level for service overlay computing. The scientific tools used are category theory and process calculi, defining software architectures and programming languages for mobile global computing systems. To make these formal approaches available for practitioners, appropriate UML extensions have been devised which provide a visual representation of the declarative modelling primitives. In addition, the process algebraic programming primitives and their mathematical theory serve for simulating and analysing UML models.

Qualitative and Quantitative Analysis Methods for Services. Mathematical models for service computing formalise different aspects of overlay computers: at this level, services are seen as abstract computational entities, modelled in a platform-independent architectural layer. The mathematical models, hidden from the developer, enable qualitative and quantitative analysis supporting the service development process and providing the means for reasoning about functional and non-functional properties of services and service aggregates. SENSORIA results include powerful mathematical analysis techniques; in particular program analysis techniques, type systems, logics, and process calculi for investigating the behaviour and the quality of service of properties of global services. These techniques are then tailored to several specific purposes: Firstly, they can be used to reveal performance bottlenecks or interactions leading to errors or violation of service contracts. Secondly, they are used to deal with security issues like confidentiality, integrity, non-interference, access control, and trust management. Finally, for critical services, deep semantic analysis may be used for certification.

Model Driven Development, Tools, and Validation. Sensoria techniques may be integrated into several software process models. Specific emphasis is placed on Model-Driven Development (MDD). Sensoria introduces automated model transformations to allow generation of formal representations from engineering models, back-translation from formal results to user-level models, and generation of code. All techniques and methods developed within Sensoria are accompanied by tools, which are integrated into a common tooling platform (the Sensoria Development Environment). To prove that the developed methods are applicable in industrial contexts, Sensoria includes several case studies from various application domains of software engineering, whose scenarios have been rigorously tested.

Summarising, the added value of Sensoria comes from the availability of sound engineering techniques supported by mathematical foundations, languages with formal semantics and associated analysis methods, and the ability to automate many of the development steps currently done by hand in the design of service-oriented software. The next three sections describe the above three points in more detail.

3 Part I: Linguistic Primitives for Modelling and Programming SOA Systems

The first theme of Sensoria has been focused on the definition of adequate linguistic primitives for modelling and programming service-oriented systems, enabling model-driven development for implementing services on different global computers. The primitives introduced in Sensoria allow both high-level system modelling as well as detailed, rigourous specifications of SOA systems using mathematical notations. Automated model transformations allow switching between these two levels, and in addition enable generation of executable code.

Sensoria has first established foundations for service description, interaction and composition, at the level of architectural specification. Based on this, core calculi for service-oriented computing have been developed, accounting for different interaction and composition architectures, like message-driven or data-driven. Finally, the core calculi have been extended to establish a solid mathematical basis for quality of service, service level agreements, workflow-like transactions with compensation, and dynamic reconfiguration.

3.1 Modelling in Service-Oriented Architectures

Modelling of Service-Oriented Architectures has been investigated on different abstraction levels and with different aims in SENSORIA, which has led to four main outcomes.

Firstly, SOA systems can be modelled on a high level of abstraction with the help of the Unified Modelling Language (UML). The UML is accepted as the *lingua franca* in the development of software systems. It is the most mature language used for modelling. However, plain UML is not expressive enough for the specification of structural and behavioural aspects of services. SENSORIA therefore provides individual UML extensions which form a *Sensoria family of profiles* for SOA development, which are jointly used to model the different aspects of service-oriented software. The SENSORIA family of profiles comprise a profile for service orchestration (*UML4SOA*), for non-functional properties of services, business policies, for implementation modes of SOAs, and service deployment. The UML extensions are further detailed in chapter *"UML Extensions for Service-Oriented Systems"*.

Secondly, one can also take a more formal approach to SOA system specification with the help of the *Sensoria Reference Modelling Language (SRML)*. SRML is inspired by the Service Component Architecture (SCA). It makes available a general assembly model and binding mechanisms for service components and clients that may have been programmed in possibly many different languages, e.g. Java, C++, BPEL, or PHP. However, where SCA supports bottom-up low-level design, SRML instead addresses top-down high-level design. More specifically, the aim was to develop models and mechanisms that support the design of complex services, and analysis techniques through which designers can verify or validate their properties. These composite services can then be put together from (heterogeneous) service components using assembly and binding techniques such as the ones provided by SCA. SRML will be discussed in detail in chapter *"The SENSORIA Reference Modelling Language"*.

Business processes typically structure their activities with workflows, which are often implemented in a rather static fashion in their IT systems. Nowadays, system requirements change rapidly as business activities try to maintain their competitive edge, and hence a predominant need arises for the IT systems to present the same agility. This problem has been investigated in SENSORIA and has lead to a new approach, *StPowla*, which marries service-oriented architecture, policies and workflows to provide businesses with this agility at execution time of their workflows. In StPowla the business is modelled as a workflow and is ultimately carried out by services. Indeed, policies provide the necessary adaptation to the varied expectations of the various business stakeholders. A key idea is that the stakeholders can define policies to adapt the core work ow by modifying the service to be invoked or the QoS levels. StPowla is further discussed in chapter *"Model-Driven Development of Adaptable Service-Oriented Business Processes"*.

Finally, another important aspect of service-oriented computing systems lies in their architecture, which must match the global structure required by the business processes they are intended to support. SENSORIA provides a solution for this problem with *Architectural Design Rewriting (ADR)*, which can be used as a formal model for architectural and business design and helps in formalising crucial aspects of the UML4SOA and SRML modelling languages mentioned above. The key features that make ADR a suitable and expressive framework are the algebraic presentation of graph-based structures, which can improve the automated support for specification, analysis and verification of service-oriented architectures and applications. ADR is discussed in chapter *"A Formal Support to Business and Architectural Design for Service-Oriented Systems"*.

3.2 Calculi for Service-Oriented Computing

SENSORIA has investigated a foundational methodology for describing service specifications and for developing a discipline for their composition. This methodology relies on services as the fundamental elements for developing applications, thus conforming to the Service-Oriented Computing (SOC) paradigm. The fundamental vehicle used in this respect has been the theory of process calculi and their operational modelling as labelled transition systems, intended as the collections of linguistic constructs, tools, models, and prototype implementations that have been developed for designing, analysing, and experimenting with open components interactions.

Core calculi have been adopted in the SENSORIA project with three main aims. First of all, they have been used to clarify and formally define the basic concepts that characterize the SENSORIA approach to the modeling of service oriented applications. In second place, they are formal models on which the SENSORIA analysis techniques have been developed. Finally, they have been used to drive the implementation of the prototypes of the SENSORIA languages for programming actual service-based systems. The SENSORIA core calculi are described in chapter *"Core Calculi for Service-Oriented Computing"*.

In a formal language, it is common to have several terms denoting the same process. To understand when different terms refer to the same process, the language needs to be equipped with a notion of *equivalence*. SENSORIA has investigated bisimilarity notions applied to some of the SENSORIA core calculi. The aim was to develop algebraic reasoning on processes by finding useful axioms (correct with respect to bisimilarity). Two different applications for this are program transformations and spatial characterizations of systems. The former is used to show how to transform object-oriented diagrams to session oriented ones, how to break sessions into smaller pieces that can be implemented using current technologies, and to show that an implementation of a service is compliant to a more abstract specification. The latter proves that bisimilarity is a congruence and shows behavioral identities that illuminate the spatial nature of processes and pave the way for establishing a normal form result. This is further discussed in chapter *"Behavioural Theory for Session-Oriented Calculi"*.

An important tool for verifying system correctness are *static analysis techniques*. Within SENSORIA, such techniques have been developed for CaSPiS (Calculus of Sessions and Pipes) and CC (Conversation Calculus), two session oriented calculi developed within the project. Each technique aims at guaranteeing a specific property one would expect from service-oriented applications. These models and techniques may be complementary used and combined in order to provide as many guarantees as possible on the correctness of services' behaviour. Chapter *"Static Analysis Techniques for Session-Oriented Calculi"* contains more information on static analysis.

A key issue of the service approach is given by its compositional nature. For example, existing services can be combined (a process which is called *orchestration*) to create a more complex business process. This yields the problem of properly selecting and configuring services to guarantee that their orchestration enjoys some desirable properties. These properties may involve functional aspects, and also non-functional aspects, like e.g. security, availability, performance, transactionality, etc. Sensoria includes a framework for designing and composing services in a *"call-by-contract" fashion*, i.e. according to their behaviour. For a discussion on how to plan compositions of services so that the resulting choreography satisfies the desired functional and non-functional properties see chapter *"Call-by-Contract for Service Discovery, Orchestration and Recovery"*.

3.3 Negotiations, Planning, and Reconfiguration

The SOC paradigm has to face several challenges like service discovery, Service Level Agreements (SLA) and Quality of Service (QoS), workflow-like transactions and compensations, monitoring and dynamic reconfiguration. Sensoria has addressed these aspects, namely SLA/QoS, transactions with compensations, and dynamic reconfiguration, to a) establish a solid mathematical basis that can serve to formalise crucial aspects of SLAs, b) distill service aggregation patterns, and c) provide a sound architectural basis for dynamic reconfigurations.

One of the ultimate goals of service-oriented computing (SOC) is to provide support for the automatic on-demand discovery of basic functionalities that, once combined, correctly compute a user defined task. To this aim, it is necessary for services to come equipped with a computer-understandable interface that exposes enough information in order to match the provided functionalities with the user needs.

Services may expose both functional properties and non-functional properties. Non-functional properties focus on the Quality of Service (QoS) and typically include performance, availability, and cost. QoS parameters play an important role in service composition and, specifically, in dynamic discovery and binding. Indeed, a service requester may have minimal QoS requirements below which a service is not considered useful. Moreover, multiple services that meet the functional requirements of a requester can still be differentiated according to their non-functional properties. In Sensoria, this challenge is addressed with a simple calculus, called *cc-pi calculus*, for modelling processes able to specify QoS requirements and to conclude QoS contracts. See chapter *"CC-Pi: A Constraint Language for Service Negotiation and Composition"* for more information.

Another prominent issue in the automated combination of services concerns the compliance between the operations invoked by the client – the client protocol – and the receive operations executed by the service – the service protocol. Among other things, this requires a) actually extracting a manageable description of the service interface (contract) from a reasonably detailed service specification, and b) guaranteeing that the services retrieved from a repository behave correctly according to the user and the other retrieved services needs. Such techniques have been investigated in Sensoria and are detailed in chapter *"Advanced Mechanisms for Service Composition, Query and Discovery"*.

In enterprise computing, services are used to model business processes, which may potentially take a large amount of time. Such activities, known as *long-running transactions*, are often supported by specialised language primitives and constructs such as exception and compensation handling. Exception handling is used to react to unexpected events, while compensation handling is used to undo previously completed activities. The impact of adding exception- and compensation handling to a language is detailed in chapter *"Advanced Mechanisms for Service Combination and Transactions"*. Furthermore, specification and refactoring of long-running transactions is discussed in chapter *"Model-Driven Development of Long-Running Transactions"*.

Finally, SOA systems should offer the ability to dynamically reconfigure SOA architectures to adapt to changing requirements. The Sensoria *Architectural Design Rewriting* approach introduced above can also be used as a foundational model for reconfigurable service-oriented systems. This is detailed in chapter *"A Formal Support to Reconfiguration of Service-Oriented Systems"*.

4 Part II: Formal Analysis of Service-Oriented Systems

The previous section has introduced language primitives and in particular core calculi for the rigourous specification of service-oriented systems. These mathematical models, hidden from the developer, enable qualitative and quantitative analysis supporting the service development process and providing the means for reasoning about functional and non-functional properties of services and service aggregates.

In Sensoria, a full spectrum of qualitative analysis techniques has been developed, thus allowing service engineers to guarantee a high level of security and trust for the location transparent delivery of services while allowing mobility of resources. Furthermore, Sensoria has investigated means for coping with the quantitative aspects of service-oriented systems. The focus lay on stochastic methods for developing analysis techniques and tools to study and verify quantitative properties such as resource usage and quality of service.

4.1 Qualitative Analysis Techniques for Service-Oriented Computing

Quality of service is becoming a key parameter in determining the success or failure of information systems offering their services using overlay computers;

techniques are needed for validating the performance of systems with respect to their specifications (in particular as regards security and trust). Sensoria has investigated analysis techniques for ensuring quality of service properties in SOC systems.

In the service-oriented environment, the view of a system can be divided into different levels; at the abstract level (as is usually the view found in academia), the system is independent of the underlying communication protocols, and at the concrete level (as is usually the view found in industry), the system must be understood in connection with how it makes use of established communication protocols. Motivated by this separation of concerns, Sensoria has devised a specification approach called *CaPiTo* to facilitate modelling systems at both the abstract and the concrete level. To bridge the gap between the two levels, an intermediary level has been defined that connects them. Chapter *"Analysing the Protocol Stack for Services"* discusses CaPiTo.

Early validation of system requirements and early detection of design errors is an important cornerstone of creating high-quality software systems. Sensoria supports such validation by a common logical framework for verifying functional properties of service-oriented systems, which has been instantiated in the *CMC* and *UMC* model checkers. The design principles used in these tools are detailed in chapter *"An Abstract, On-The-Fly Framework for the Verification of Service Oriented Systems"*.

Service architectures should be dynamic, where service bindings and contexts change with the environment. The task of designing and analysing such architectures becomes very complex. As a remedy, Sensoria has developed a specification profile and analysis framework for so-called *service modes*. A service mode provides an encapsulation of both specification and adaptation in different service scenarios. The modes approach is described in chapter *"Specification and Analysis of Dynamically-Reconfigurable Service Architectures"*.

Finally, Sensoria also provides various tools which accompany the formal methods laid out in the previous sections for analysis and verification of service oriented systems. Four of these tools (CMC, UMC, ChorSLMC, and LocUsT) are described in chapter *"Tools and Verification"*.

4.2 Quantitative Analysis Techniques for Service-Oriented Computing

Service-oriented computing is made up of many activities that are quantity driven (service level agreement, quality of service, negotiation, orchestration, resource usage are only some of them). As a consequence, the issue of quantitative description and analysis of such systems in the global computing setting needs to be addressed in all the phases of SOA software development.

Service-oriented computing goes beyond the usual problems encountered in network based systems by its focus on open-endedness. Addressing key functional aspects of network aware programming such as distribution awareness, mobility and security, and guaranteeing their integration with performance and dependability guarantees is complemented in Sensoria by specific features intended for service-oriented architectures. Chapter *"SoSL: Service Oriented Stochastic logics"* introduces the temporal logic *SoSL* and shows how SoSL formulae can be model-checked against systems descriptions expressed with *MarCaSPiS*, a process calculus designed for addressing quantitative aspects of SOC.

Service Level Agreements (SLAs) underpin the expectation of the performance of a system as seen by a particular client of a service. Most often an SLA will speak about the response-time of the system. In general, a qualitative analysis of a service-oriented system requires abstracting the parts of the system which are relevant to the analysis into an abstract model which is generally defined in some modelling formalism. In SENSORIA, the process algebra *PEPA* was used for this purpose, along with *eXtended Stochastic Probes (XSP)* for the specification of the passage of interest within the defined model. Both are detailed in chapter *"Evaluating Service Level Agreements using Observational Probes"*.

The quantitative analysis of large-scale applications using discrete-state models is fundamentally hampered by the rapid growth of the state space as a function of the number of components in the system (state-space explosion). However, service-oriented architectures often lie in the large-scale section of the application landscape. Here, the fluid-flow interpretation of the stochastic process calculus PEPA provides a very useful tool for the performance evaluation of large-scale systems because the tractability of the numerical solution does not depend upon the population levels of the system under study. Scaling performance analysis using fluid-flow approximation is discussed in chapter *"Scaling Performance Analysis using Fluid-Flow Approximation"*; a related method called *passage-end calculations* is discussed in chapter *"Passage-End Analysis for Analysing Robot Movement"*.

Finally, as in the previous section, SENSORIA provides various tools performing quantitative analysis accompanying the methods and techniques laid out in above. Chapter *"Quantitative Analysis of Services"* shows instances of exact model checking of MarCaSPiS against the both state-aware and action-aware logic SoSL, exact and statistical model checking of sCOWS against the state-aware logic CSL, and querying of PEPA models by terms of the XSP language that expresses both state-aware and action-aware stochastic probes.

5 Part III: Model-Driven Development, Tools, and Validation

The third theme of the SENSORIA project deals with the engineering aspects of service-oriented system construction. It builds on both previous themes and introduces the glue with brings all SENSORIA methods and tools together and wraps them in a validation cocoon.

The theme includes new model-based development techniques for refining and transforming service specifications, novel techniques for deploying service descriptions on different global computers, and methods for reengineering legacy systems into service-oriented ones. These results form the cornerstone for the practical usability of the Sensoria approach, and are complemented by realistic case studies from industrial partners.

5.1 Model-Driven Development and Reverse-Engineering for Service-Oriented Systems

In order to allow developers to use the qualitative and quantitative analysis of services discussed in previous sections, Sensoria has created model transformation tools and concrete transformations for a) strengthening and streamlining the connections between formal languages and high-level systems models, b) orchestrating the experimentation and analysis process, and c) deploying well-managed analysis on systems of increasing scale and complexity. Due to their complexity, SOA systems also require advanced and diverse mechanisms for deployment and runtime management. Sensoria has investigated such mechanisms and provides a suite of tools and techniques for transforming SOA artefacts, deploying service-oriented systems on global computers and for reengineering legacy systems into services.

Model transformation serves as a key technology for the model-driven service engineering approach suggested by Sensoria. To be effective for a day-by-day use in the engineering process, Sensoria had to solve some common problems (traceability, back-annotation, intuitive requirement definition, etc.) with model transformation techniques. These problems, and the solutions provided by Sensoria, are introduced in chapter *"Methodologies for Model-Driven Development and Deployment: an Overview"*, along with an end-to-end example for using model-driven techniques for analysing services, and the tool support provided by the project. Going into more detail, Sensoria has employed model transformation techniques which make use of precise mathematical foundations provided by the paradigm of graph transformation. The unique challenges and solutions provided by the graph transformation paradigm are discussed in chapter *"Advances in Model Transformations by Graph Transformation: Specification, Analysis and Execution"*.

The ability to dynamically compose autonomous services for meeting the requirements of different applications is one of the major advantages offered by the service-oriented computing paradigm. A dynamic service composition implies that services requesters can be dynamically bound to most appropriate service providers that are currently available, in order to optimally satisfy the service requirements. At the same time, the autonomy of services involved in a composition means that the resulting composition may need to be adapted in response to changes in the service capabilities or requirements. Naturally, the infrastructure and technologies for providing runtime support for dynamic and adaptive composition of services form the backbone of the above process. Within Sensoria, the *Dino* approach has been developed, which provides comprehensive support for all stages of a service composition life-cycle, namely: service discovery, selection, binding, delivery, monitoring and adaptation. More on Dino can be found in chapter *"Runtime Support for Dynamic and Adaptive Service Composition"*.

Many companies and organisations have already invested heavily in their IT infrastructure in the past. The migration of such legacy systems towards service-oriented architectures is therefore of particular importance. Sensoria has developed a general methodology for *software reengineering*. This method has been instantiated to allow service components to be extracted from legacy applications. Chapter *"Legacy Transformations for Extracting Service Components"* describes a systematic way of addressing such reengineering projects with a high degree of automation while being largely independent of the programming language.

Developing service-oriented software involves dealing with multiple languages, platforms, artefacts, and tools. The tasks carried out during development are varied, ranging from modelling to implementation, from analysis to testing. For many of these tasks, the Sensoria project has provided tools aiding developers in their work. To enable developers to find, use, and combine these tools, Sensoria provides a tool integration platform, the Sensoria *Development Environment (SDE)*, which a) gives an overview of available tools and their area of application, b) allows developers to use tools in a homogeneous way, re-arranging tool functionality as required, and c) enables users to stay on a chosen level of abstraction, hiding formal details as much as possible. The SDE is described in detail in chapter *"The* Sensoria *Development Environment"*.

5.2 Case Studies and Patterns

The research in the Sensoria project has been based on a series of realistic case studies, which have been used for feeding and steering the research process, discussing and communicating ideas among partners, and finally disseminating research results to and getting feedback from the research community at large, both in industry and academia.

The immediately following chapter *"Introduction to the* Sensoria *Case Studies"* will introduce the *case studies* used within the project. Having in mind the relevance that these areas have in society and the economy, three case studies have been extensively used in Sensoria during the whole project. Two of the case studies come from industrial applications in automotive, telecommunication and finance domains, and one comes from an academic application for distributed e-learning and course management. After the Sensoria notations have been introduced, chapter *"Specification and Implementation of Demonstrators for the Case Studies"* goes into more detail about the case studies.

Chapter "Sensoria *Results Applied to the Case Studies"* provides a *concise overview* of the exploitation of Sensoria results in all of our case studies. In addition to this overview, chapter *"Analysing Robot Movement Using the* Sensoria *Methods"* provides an in-depth review of Sensoria methods applied to the *robot case study*, while chapter *"The* Sensoria *Approach Applied to the Finance Case Study"* provides an in-depth view of the Sensoria approach applied to the *finance case study*.

Finally, the Sensoria project contributes to existing software development processes and methodologies by providing *patterns* for common tasks and problems encountered when developing service-oriented software systems. Chapter *"*SENSORIA *Development Patterns"* discusses a pattern language for augmenting service engineering with formal analysis, transformation and dynamicity. The pattern language is designed to help software developers choose appropriate tools and techniques to develop service-oriented systems with support from formal methods; the full pattern catalog spans the whole development process, from the modeling stage to deployment activities. Chapter *"Organizational Patterns for Security and Dependability: From Design to Application"* specifically discusses security and dependability (S&D) patterns, which are of great help to designers when developing service-oriented software systems.

6 Conclusion

In todays networked world, service-oriented computing and service-oriented architectures are an accepted architectural style for creating massively distributed network-based applications. The SENSORIA project has contributed to this field by addressing the foundations of modelling, transforming, analysing, and deploying service-oriented artifacts as well as reengineering legacy systems. SENSORIA has thereby created a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach.

SENSORIA has brought mathematically well-founded modelling technology within the reach of service-oriented software designers and developers. By using these techniques and tools, IT-dependent organisations can move to a higher and more mature level of SOA software development. In particular, using the SENSORIA techniques can increase the quality of SOA applications, measured both in qualitative and quantitative terms. As SENSORIA methods are portable to existing platforms, application of these methods is possible while keeping existing investments.

Acknowledgements. This work has been supported by the EU FET-GC2 IP project SENSORIA (IST-2005-016004).