Methodologies for Model-Driven Development and Deployment: an Overview

László Gönczy, Ábel Hegedüs, and Dániel Varró

Budapest University of Technology and Economics, Hungary
{gonczy,hegedusa,varro}@mit.bme.hu

1 Introduction

This chapter introduces a model-driven SENSORIA approach for service engineering. The project delivered a comprehensive approach for service modeling, analysis and deployment with novel *modeling languages*, qualitative and quantitative techniques for service *analysis*, automated model driven *deplyoment* mechanisms and legacy transformations. *Model transformation* served as a key technology for model-driven service engineering. The first part of the chapter first discusses the overall methodology and then briefly overviews some key contributions of the project (Sec. 2). Obviously, these individual contributions are presented from the viewpoint of model-driven development. Most of these are discussed in detail in other chapters of this book or in publications related to the project.

The second part (Sec. 3) presents a selected "end-to-end" example for using model-driven techniques for analysing services. Here high-level, standard models of business processes and their correctness requirements are translated to a formal model (namely, transition systems and temporal logic formulae) in order to enable exhaustive verification by a model checker, which is a common scenario in the SENSORIA project.

Furthermore, this forward model transformation is also complemented with the back-annotation of analysis results to the original service models. The technique we present enables the easy visualization/simulation of model checker results right on the original business processes, therefore enabling the service developer to correct design flaws.

The tool support integrated into the SENSORIA Development Environment is briefly discussed in Sec. 3.5. Finally, Section 4 discusses related work and Section 5 concludes the paper.

2 Overview on Model-Driven Methodologies

2.1 The Sensoria Service Engineering Approach

This crosscutting chapter presents the *engineering vision* of the SENSORIA project, which facilitates a model-driven development approach. After a brief conceptual introduction, the chapter presents a high-level overview in order to demonstrate the feasibility of the approach by summarizing selected achievements in the project from a practical, engineering and tool-oriented viewpoint.

Actors in a service-oriented project A primary goal of the SENSORIA project is to provide support for different stakeholders and actors during the entire project lifecycle for developing service-oriented overlay systems of a justifiable quality. These participants inevitably include the following ones:

- *Domain experts* are responsible for synthesizing requirements from business-related knowledge such as organization-specific roles, typical business scenarios or workflows, or business-critical data. While domain experts are obviously experts in their own application domain, they typically lack general software (and service) engineering skills, thus high-level, easy-to-understand languages are essential for them to record their business knowledge.
- *Service modelers* are in charge of the technical design of service-oriented systems which has to meet the business-related requirements. Service modelers are typically engineers with skills in modern service-oriented modeling languages and design technologies. However, they are typically less knowledgeable in how to provide guarantees for the proven quality of service.
- *Service certifiers* are frequently a project-independent entity or authority being responsible for assuring the approved quality of services. While today, this role is still restricted to dedicated application areas (such as mission or safety-critical applications), it is expected that the role of dependability (i.e. justifiable quality of services) will drastically increase in traditional business areas. Service certifiers are typically skilled in formal (mathematical)analysis and testing techniques in order to carry out precise analysis of the service which is currently in the design phase.
- *Service managers* are in charge of the proper deployment and maintenance (e.g. upgrade) of business-critical services. They are experts in the underlying service infrastructures.

SENSORIA proposes a model-driven approach for the entire development cycle of services based applications and infrastructures including the design, the formal analysis, the deployment and re-engineering of services. The core ideas of the SENSORIA engineering approach are illustrated in Fig. 1.

The main contributions of SENSORIA can be summarized from an engineering perspective as follows:

- Precise capturing of domain-specific requirements
- High-level front-end service modeling languages
- Hidden formal analysis of services
- Deep semantic analysis for certification
- Automated deployment of services to various infrastructures
- Customizable orchestrator for tool integration
- Model transformations for bridging models and languages
- Reengineering of legacy services
- Standards-compliant languages and service infrastructure

The current paper provides a brief, high-level overview of selected contributions within each category above.

Experience on using these methods on SENSORIA case studies is collected in Chapter 7-2.

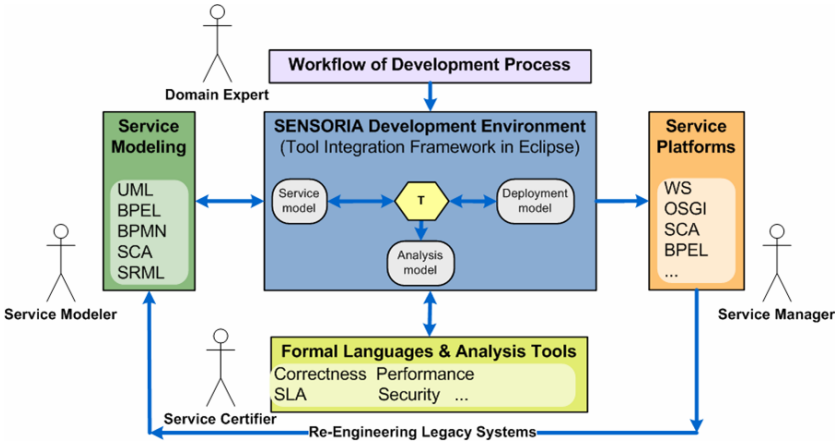

Fig. 1. The SENSORIAengineering approach

2.2 Contributions of Sensoria

Precise capturing of domain-specific requirements This work is an enhancement of the Requirement Engineering technique presented previously. Here the emphasis was on Business Process Reengineering where functional and security requirements must be guaranteed during the engineering process. Thus the connections between business processes and requirement models were investigated and the notion of *goal equivalence* was introduced. A framework was defined to support *Goal Equivalent Secure Business Process Reengineering* in [17].

High-level front-end service modeling languages. In order to support service modelers, SENSORIA facilitates the use of high-level front-end service modeling languages. A primary means for that is the definition of a service-oriented UML Profile, thus off-the-shelf UML CASE tools can be used by service engineers to construct service models. UML4SOA is described in detail in Chapter 1-1.

As an alternate solution, SENSORIA proposes a new domain-specific modeling language (called SRML) for a rich semantic definition of components and services. A visual SRML editor, and an EMF-compliant SRML metamodel is developed, the approach is reported in detail in Chapter 1-2.

Hidden formal analysis of services As a core contribution, SENSORIA facilitates a model-driven, hidden formal analysis of service-oriented overlay systems. Since service modelers typically lack mathematical skills to carry out an in-depth analysis of the system in an early phase of design, we carry out automated model analysis by transforming high-level service models into precise mathematical models in order to carry out quantitative and qualitative analysis. Results of the

mathematical analysis are aimed to be back-annotated to the high-level models service engineers, thus hiding the technicalities of the underlying mathematical analysis.

A method was created for *model-based qualitative analysis of services via call by contract*. Here UML models are passed to a static checker by automated model transformations in order to analyze service behaviour against security policies [20].

Techniques for *model-based quantitative performance analysis of services* by extracting a performance model from high-level service models captured in UML were also investigated. The performance model is mapped onto a stochastic process of the PEPA framework whose analysis allows the service modeller to obtain quantitative measures such as throughput and utilisation (Chapter 5-2 and Chapter 5-3).

We proposed the use of Modes to abstract a selected set of services, and use UML2 models to analyse self-managing and reconfigurable service architectures for consistency and constraints. In addition, coordination processes may be synthesised to manage the changes in architecture as environmental changes occur. The approach is illustrated through the use of the SENSORIA Development Enrivonemnt with collaborating UML2, Darwin and Ponder2 model-transformation to deployment artefacts (see Chapter 4-4).

Recently, we developed novel model-based *analysis methods for service orchestration* designed in UML4SOA. Here we check consistency and protocol conformance of service compositions described as UML activity diagrams and protocols defined by UML state machines. Details of this method are provided in [18].

Finally, we also help service developers to estimate the cost of reliability (in terms of response time overhead) by introducing performability analysis techniques for reliable messaging middleware. UML4SOA models with specifications on non-functional parameters of service communication are transformed into PEPA models according to messaging mode/characteristics and the cost of middleware configuration alternatives can be evaluated by sensitivity analysis (see [11] for details).

Deep semantic analysis for certification. The service-oriented calculi developed in WP2 are intended for deep semantic analysis carried out by service certifiers having in-depth mathematical knowledge. Previously we also demonstrated that a model-driven approach is also applicable here as well by bridging UML models of service orchestrations and sound formal notations. More specifically, a transformation is presented which maps UML activity diagrams to the saga calculus. This allows a sound formal analysis of the orchestration's control flow based on a simple and formally easily amenable representation.

A transformation has also been developed to map UML4OSA models to Jolie orchestrations which permits the use of the analysis features provided by SOCK, the formal background of Jolie [13]. This transformation is part of the MDD4SOA toolkit.

Automated deployment of services to standards-compliant service infrastructures Service managers responsible for deploying and maintaining service-oriented applications need to face the challenge that more and more emphasis is put on the reliability, availability, security, etc. of such services. In order to meet such non-functional requirements, a service needs to be designed for reliability by making design decisions on a high, architectural level. Details of the techniques are presented in [12], where a model-driven approach for the automated deployment of services to standards-compliant service infrastructures is described. Starting from a platform-independent service model enriched by non-functional attributes for reliable messaging, low-level service configuration descriptors are generated by appropriate model transformations for standards-compliant middleware supporting reliable and secure messaging. This year we extended existing model transformations to include the Apache Axis2 platform as target execution environment.

As a "side effect" of these developments, a novel approach is being investigated to facilitate the development of *customizable model transformations*. As illustrated by the above techniques, there are multiple model transformations during analysis and development of service oriented systems. In these transformations, typically there are steps where a high level system model is parsed in order to generate a subset of the model which is relevant for a particular analysis method (performance, security, etc) while other steps aim at the creation of deployment artifacts for different platforms (e.g. Web service implementation on Apache/IBM, considering different configuration constraints). These steps all need similar model transformations/translations which can be "parametrized" by high level engineering models. This approach needs the development of models rather than transformations.

Customizable orchestrator for tool integration. Different application domains and organizations frequently need to customize the overall development process to the specific needs of the domain. However, when new tools are intended to be added to the workflow of the organization, this requires significant efforts in tool integration. In order to bridge the gap between the development process and the development tools (thus reducing the complexity of tool integration), the SENSORIA Development Environment (SDE) is provided, acting as a central orchestrator for individual tools and services. SDE uses Eclipse based de facto standards, such as EMF-based interfaces for models and OSGI services for design, analysis, model transformation and deployment steps.

Advances on the SDE and currently integrated tools are described in Chapter 6-5.

Model transformations for bridging models and languages. The model-driven development, analysis and deployment of services with justifiable quality necessitates that the automated model transformations bridging different languages and tools are precise themselves. SENSORIA builds on modern Eclipse-based model transformation frameworks (such as VIATRA2 and the TIGER

EMF Transformer) supporting standards EMF-based interfaces with precise mathematical foundations provided by the paradigm of graph transformation.

Furthermore, we have also investigated innovative ways for accelerating the process of designing model transformations. Model transformation by example is a novel approach in model-driven software engineering to derive model transformation rules from an initial prototypical set of interrelated source and target models, which describe critical cases of the model transformation problem in a purely declarative way.

Recently, we investigated techniques for *incremental, live model transformations*. We adapted the well-known RETE algorithm from the field of rule-based systems to make pattern matching more efficient, therefore reducing execution time of model transformations. We implemented the algorithm for the VIATRA2 model transformation framework. Unlike batch transformations, live transformations are triggered by model changes and they are executed incrementally to synchronize models.

Furthermore, the Moment-GT model transformation engine has been developed to facilitate verifiable model transformations using the Maude rewriting logic engine as formal background. These techniques are presented in Chapter 6-2).

Reengineering of legacy services In order to support the reengineering and redeployment of existing legacy applications to more modern service-oriented platforms, the reengineering methodology is summarized from the global software engineering view of SENSORIA.Each instantiation of the methodology determines what programming language can be used as input and what concrete platform will the services adhere to. Reengineering is also carried out by model transformations (Chapter 6-4).

Standards-compliant languages and service infrastructure In order to widen the practical usability of results, interfaces and platforms used within the SENSORIA engineering approach are compliant with standards and/or industrial best practices. For instance, service models are captured in UML, BPEL or using EMF-compliant graphical editors in Eclipse. Target deployment platforms include service infrastructures supporting various WS-* standards on several (IBM, Apache) platforms. Services are deployed by using standard service descriptors such as WSDL, and currently there is an ongoing development to support SCA.

3 From BPEL to SAL and Back: an End-to-End Example on Model-Driven Analysis

The development of a fully-fledged verification tool directed by model-driven analysis necessitates the application of numerous model-based techniques. In

this section these techniques are presented through a complex end-to-end example implementation providing design-time verification support for business processes. First the verification approach is described, followed by a short presentation of the challenges in model-driven analysis. Finally the usage of the different techniques are described on the example implementation.

3.1 Practical Design-time Verification of Business Processes

Motivation Business processes are often used to coordinate the work of different stakeholders in business-to-business collaborations as well as Enterprise Application Integration. Since these workflows set up the cooperation between actors, their quality is critical to the organization and any malfunction may have a significant negative impact on financial aspects. To minimize the possibility of failures, designers and analysts need powerful tools to guarantee the correctness of business workflows.

Approach The main steps of the method presented in [16] are illustrated in Fig. 2. In the current chapter, we restrict our investigations to using BPEL as an input language. However, the SENSORIA toolset offers the higher-level UML4SOA models to capture business processes and derive actual BPEL descriptions by automated model transformation. In the next step, the input BPEL business process description is transformed into a formal model in the form of state transition systems. In the second stage, this transition system is projected into the language of the Symbolic Analysis Laboratory (SAL) [25]. The actual verification is then carried out with symbolic or bounded model checking techniques [26]. Requirements against the business process are captured as the expressions of the Linear Temporal Logic [5]. General (application-independent) requirements and arbitrary business process-specific requirements may be verified with the model checking technique. As a distinctive feature of this verification technique, it provides support for analysing error propagation between variables.

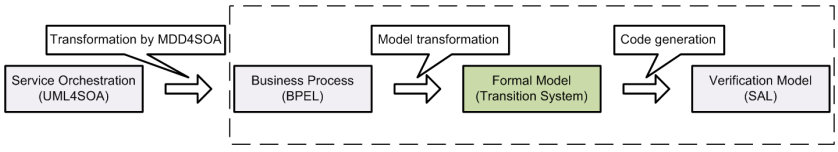

Fig. 2. BPEL Verification approach overview

In the method, model checking is used for verification purposes. The result of model checking is a sequence of actions which violate the requirement (counter-example). The system satisfies the requirement if a counter-example cannot be found. The counter-example represents an execution of the BPEL process, but deriving and presenting this execution is non-trivial.

Running example The SENSORIA project incorporates complex case studies from different domains which are used for demonstration purposes. We selected the Finance Case Study [2] as a running example for our paper. The case study includes a credit request process which we modeled in BPEL, a simplified version of the process is shown on Fig. 3.

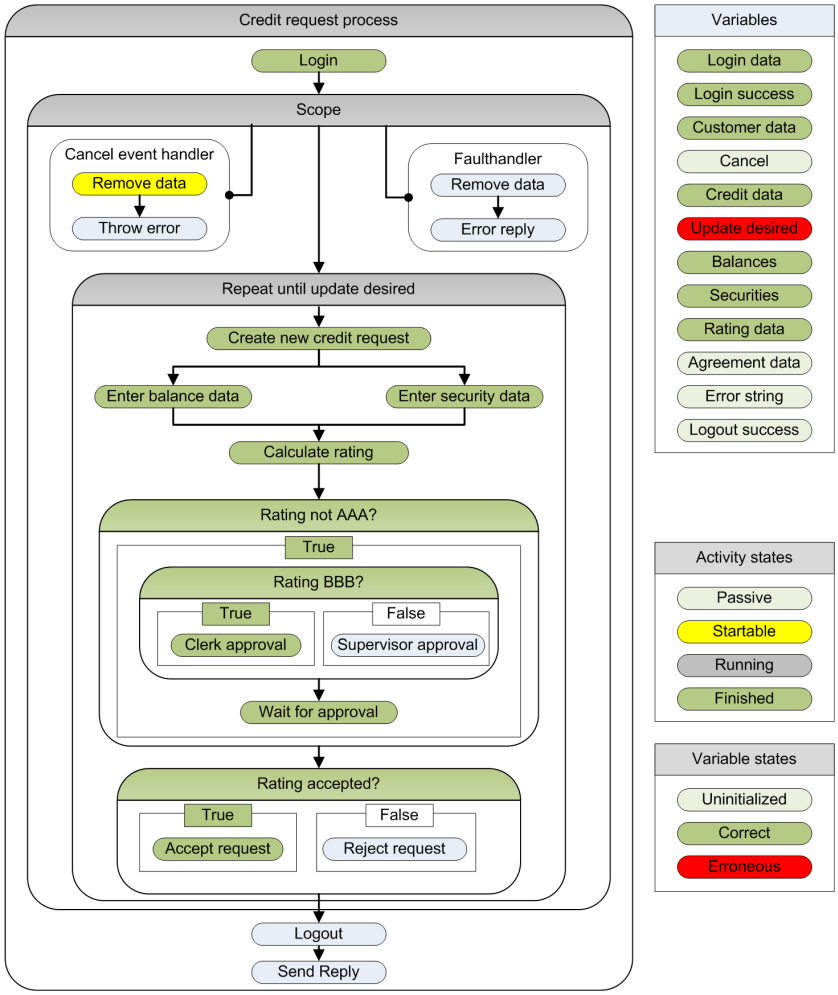

Fig. 3. Credit request BPEL process

The credit request process starts with a `Login` part where the client logs in the system, if the login is successful, the main part (`Scope`) of the process starts, enabling an`Event handler` executed if the process is canceled and a `Fault handler` for catching errors. Next a cycle is started (`Repeat until`) which re-

peats as long as the request is not accepted and updates are made. The cycle core starts with `creating a new request`, followed by entering the `Balance` and `Security` data and `Calculating the rating`. If the rating is *AAA* the rating is accepted at once, otherwise *BBB* ratings are approved by a clerk, the rest are approved by a supervisor. After the approval is returned (`Wait for approval`), the request is accepted if the rating was accepted (`Rating accepted?`). If the request is rejected (`Reject request`), the client can update the request and try again. Finally the process finishes after `logging out`.

Verification example The implemented running example business process was verified using general requirements. Fig. 3 illustrates a snapshot from the animation of the execution. The validated requirement stated that the `Update desired` variable is always written during the execution before reading (i.e. no uninitialized reading occurs). Verification revealed that the process does not satisfy this requirement. Specifically, this variable is only written if the request is rejected and the client wishes to update some of the data to try again (during the `Reject request` part). Therefore, when the request is accepted on the first try (`Accept request`), the evaluation of the condition for the `Repeat until` activity leads to an uninitialized variable reading error. This minor error can be corrected multiple ways including: initializing the variable at the beginning of the process or updating the variable when accepting the request.

3.2 Methodological Overview

The SENSORIA engineering approach (see Sec. 2.1) includes the usage of hidden formal methods. A more detailed view of this part of the approach is presented through the BPEL verification method.

The general overview of model-driven design is shown on Fig. 4. The high-level system models are used to create formal models by model transformation. The definition of the syntax of the models are created by metamodeling. Model importers are defined for creating instance models conforming to metamodels from external data and code generation is employed for exporting the generated formal model for external analysis tools. Traceability information created during the transformation is used for defining requirements verified during analysis and for aiding the back-annotation of the analysis results to the high-level system model.

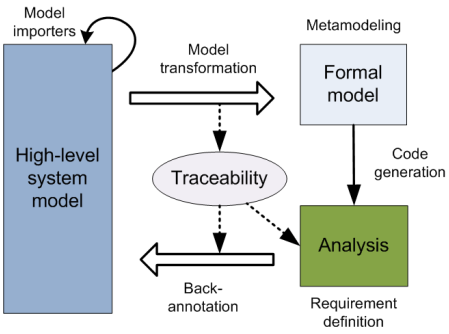

Fig. 4. Methodological overview

3.3 Challenges in Model-Driven Analysis

Automated model transformations are widely used for creating analysis models from design models for executing validation, verification, qualitative or quantitative analysis. Although numerous approaches were defined for analyzable model generation, they often lack solutions for processing analysis results automatically and thus fail to present them to the analyst in an intuitive, design model level representation. This reverse transformation problem, called *back-annotation* is non-trivial and appears as an additional challenge in most analysis techniques.

Storing the information regarding the correspondence of the design and analysis models is strongly related to back-annotation. *Traceability* is a common requirement in software development and specifically in model transformations. Several aspects of model-based design necessitate traceability information for either automatic execution or aiding user interaction. These aspects include multiple-phased transformations, requirement definition and back-annotation.

Defining requirements for analysis models is a core aspect for numerous analysis techniques. In order to ensure that formal methods remain hidden through the approach, *intuitive requirement definition* support is essential. Instead of assembling a formula manually, the user should be able to define them on a graphical user interface where general requirements can be parameterized based on the actual model and domain-specific ones can be assembled using domain-specific terms.

3.4 Implementation

The implementation of the method requires the application of a combination of MDA techniques. Note that it is possible to start out from a UML4SOA service orchestration by generating the business process with the MDD4SOA toolkit. In order to be able manipulate model instances, first the metamodels for both the BPEL business processes (source) and the transition systems (target) are created using the metamodeling capabilities of VIATRA. The source model instances are created from the XML format process description files using an importer. The target model is then constructed from the source model with the execution of an automatic model transformation. The transition system description (SAL model) is created using a special transformation which implements code generation. The analysis is carried out by model checking requirements defined against the business process on the generated model. Back-annotation of the model checking results is provided by another transformation which uses the traceability information generated during the source-target transformation. The various tools and their input-outputs are illustrated in Fig. 5.

Metamodeling Metamodels for the source and target models are created using the VIATRA Textual Metamodeling Language (VTML). VTML is capable of describing arbitrary model structures including type and containment hierarchy, instantiation and user defined relations. The BPEL metamodel is systematically

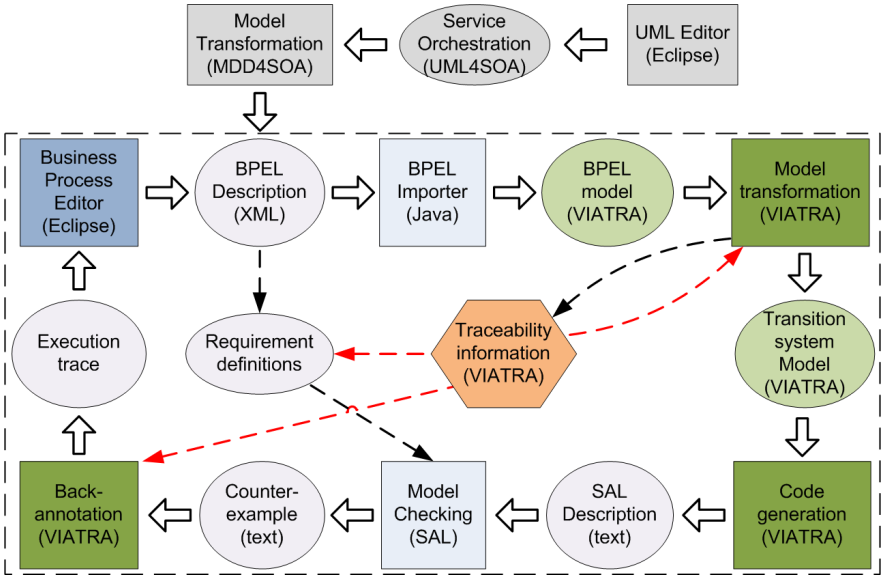

Fig. 5. BPEL Verification approach steps

created using the XML schema definition provided with the standard. The transition systems metamodel is also systematically created, based on the official SAL DTD.

Traceability An additional supporting metamodel is used during the BPEL2SAL transformation for *storing the traceability information* between the source and target model instance elements (see Fig. 5). It is also called the *static traceability metamodel*, which is used for representing the correspondence between the structure of models. The metamodel is separated from both the source or target models to ensure that arbitrary formal models can be utilized for analysis purposes without having to change the source metamodel. The elements of the metamodel are called *traceability records*.

The results of the analysis represent an execution trace of the business process described as a counter-example in the transition system. Note that the metamodels defined for BPEL and SAL describe only the structure of the language while an execution trace contains both runtime and history information as well. Therefore additional metamodels are required for supporting back-annotation (see Fig. 5).

Counter-example and execution traces The SAL counter-example is modeled by taking into account two aspects of the language. First the actual state of the transition system has to be modeled by storing *runtime information* such as the values of the variables and the states of the transitions. Then a *execution trace metamodel* is defined which is capable of storing the history of changes

in the runtime model. Similarly, the same kind of metamodels are defined for BPEL representing the actual state of the executed process instance and the changing of the state due to execution. Furthermore, as the steps of the counter-example correspond to the BPEL execution, the correspondence between the behavior of the models is stored by connecting the steps in the source and target trace models. This is called the *dynamic traceability model*. Note that from the back-annotation point of view, BPEL process is the source static model of the BPEL2SAL transformation while the BPEL trace model is the target of the SAL2BPEL transformation.

Model import The static metamodels define an abstract syntax for BPEL processes that can be used for creating instance models. However these instance models have to be created from the BPEL process description (concrete syntax). Although possible, manual model creation is strongly discouraged due to high error-probability.

Instead, *importers* are implemented to provide support for creating source models automatically from the business process description XML files. The importer utilizes that the metamodel corresponds to the schema of the XML file and creates the model instance using a *generic solution* by retrieving the metamodel elements based on the type of the currently parsed XML element.

It is important to note that BPEL processes created manually can also contain unintentional syntax errors. The MDD4SOA toolset can be used for generating the BPEL description from higher-level UML4SOA models, thus eliminating the possibility of such errors.

Model transformation The *BPEL2SAL transformation* is the main component of the implementation which constructs the target model by traversing the source model using complex model transformation rules and patterns. Although BPEL includes numerous element types which have different semantics it is possible to extract several generic rules to decrease the complexity of the transformation program.

The SAL transition system has separate parts which are generated at different phases of the transformation. Although the traceability records are created when the *variable declarations* are handled, the variables are needed in both the *variable initialization* and *transition construction* phases. Thus the traceability model is used repeatedly to find the corresponding variables to the relevant BPEL elements (e.g. find the SAL element corresponding to the BPEL variable `Balances` which is written during the execution of the `Enter balance data` activity). Note that generally transformations can be separated to various phases thus this scenario appears in many cases.

Model export The model generated by the BPEL2SAL transformation cannot be used for verification as it is. Similarly to the importers, an automatic solution is needed for exporting abstract models to their concrete syntax.

Code generation is used to export the target model to a file in the appropriate format in order to be verifiable by the SAL model checking framework. This transformation traverses the target model using simple rules and patterns which correspond to the grammar of SAL described in its Data Type Definition.

Requirement definition The requirements against business process are defined as LTL formulae evaluated through model checking. Note that the requirements which are *validated against the business process* can be best described using the source model (i.e. the BPEL process itself [30]). However, for realization they have to be *formally specified as an LTL formula* using the formalism of the target model (in this case, the SAL transition system variables). The traceability model can be used to identify which SAL variable should be used for a given BPEL element (e.g. to describe that the `Login` activity always finishes the corresponding SAL variable is needed).

User-guided definition General requirement patterns can be defined by using the actual BPEL process as a parameter. By creating general requirement templates the effort required to assemble an LTL formula can be removed. For example a requirement template can be the following: $G(< variable > / = onlyRead)$ where $< variable >$ is the parameter selected from the variables of the verifiable BPEL process. Such templates are used to provide a user interface for requirement definition.

Syntax checking LTL formulae can be tailored to express arbitrary requirements, however it is easy to make syntax errors when this is done manually. Although such errors are recognized by well-formedness checks in the model checking framework, it is advantageous to validate the formula before initializing the framework. A formula parser integrated into the user interface and implementing the grammar of LTL gives instant feedback by pointing out which part of the formula is grammatically incorrect.

Further improvement possibilities The definition of process-specific requirements has two aspects for which integrated user interface support would be essential. One is the automatic translation of the BPEL process elements to transition system variable names by selecting them in the graphical process developer interface. The other is the introduction of intuitive BPEL-specific requirement building blocks, such as *"happens after"*, *"is in given state"*, *"happens once/never in all/at least one execution"*. The combination of these techniques could result in an effective requirement definition interface.

From counter-example to trace model The SAL trace model is created automatically from the plain text counter-example returned by the model checking framework. The implementation takes advantage of the built-in generic EMF metamodel and importer of the VIATRA framework by generating first an EMF

model for the transition system and the counter-example which can be imported into the VIATRA framework.

The imported VIATRA EMF model conforms to the EMF metamodel and contains abundant information unnecessary for the back-annotation. Therefore a preprocessing transformation is used for generating the domain-specific SAL trace model from the EMF model. This trace model conforms to the SAL simulation trace metamodel.

Back-annotation transformation The execution of the business process is represented with a BPEL trace model generated from the SAL trace model by the back-annotation transformation. The back-annotation transformation is implemented as an interface in the sense that it provides the following functions for handling the BPEL trace: (1) *initialize* for creating the dynamic BPEL model and the empty trace; (2) *forward step* for updating the dynamic model according to the next step in the trace, the step is generated based on the corresponding SAL trace step if it does not exist yet; (3) *backwards step* for reverting the dynamic model to the state before the actual step; and (4) *reset* for returning to the start of the trace and the initial state of the dynamic model. After the execution of each function, the state changes of the BPEL elements are exported so that they can be used outside the model transformation framework (e.g. to drive the animation of the BPEL process).

The transformation uses the traceability information (a model instance of the static traceability metamodel which is one of the assisting metamodels) generated during the BPEL2SAL transformation. Furthermore dynamic traceability information is created whenever a forward step requires the generation of a BPEL step. This information is used for identifying the step in the SAL trace model from which the transformation has to continue.

Exporting model changes The changes in dynamic model are exported for driving the animation of the business process execution. The changes are described as a pair containing the fully qualified name of the element and its new state. Exporting is implemented by defining a new function for the VIATRA framework which can be used during model transformation.

As the changes have to be stored in memory until the transformation finishes, a service-based *export manager* is implemented that provides two service operations. The first can be used from the transformation function for storing the changes, the second is used for retrieving the changes. By implementing the export manager as a service, the back-annotation transformation and the tool used for presenting the BPEL execution are completely separated and either can be replaced by alternative techniques.

Summary In this section the challenges and techniques related to model-driven design were described through an end-to-end example providing design-time verification for business processes. The Finance case study is used to illustrate how

the verification results are represented with the animation of the BPEL process execution. Among the challenges of model-driven analysis, back-annotation, traceability and requirement definition were described. The model-driven techniques present in the example were detailed from the implementation point of view.

3.5 Overview of integrated toolchain

The implemented development tool provides complex functionality including a user interface for verifying business processes with general and user defined requirements and an extension for the Eclipse BPEL Designer [1] graphical business process developer tool. This extension is capable of animating the business process execution derived from the counter-example returned by the model checker. This high-level tool depends on several other tools which are integrated in order to hide from the user the technical details and the formal methods used.

Tool-integration Using the SENSORIA *Development Environment* (SDE), the low-level analysis and transformation tools are completely separated from the high-level tool. Fig. 6 shows how the different tools are connected to form the complete verification assistant tool. The functionality of both the VIATRA *framework* and the *SAL framework* is available through integration tools developed for the SDE. The *BPEL2SAL Tool* implements functions corresponding to the steps of the method such as transforming a business process, checking certain requirements and exporting the verification results. These tools provide their functionality as services through the SDE.

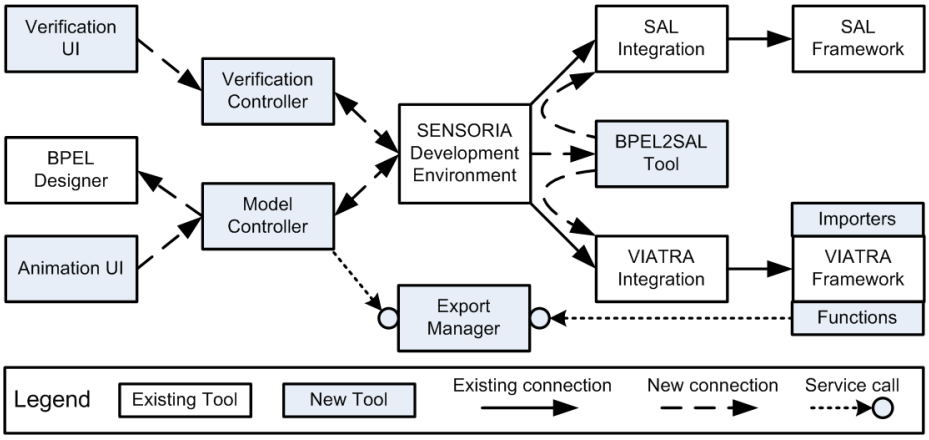

Fig. 6. Integrated tools overview

The verification is carried out by the user through the *Verification User Interface* integrated into the Eclipse framework. The graphical interface is separated

from the *Verification Controller* which contains the business logic for the verification tool. It performs the selected operation by first checking the acceptability of the parameters then calling the service of the *BPEL2SAL Tool*, finally the results are displayed on the interface.

The back-annotation is implemented as animation of the BPEL process execution in the *BPEL Designer*. An *Animation User Interface* is used for selecting an exported verification result and controlling the animation. The business logic for this interface is implemented in the *Model Controller* that is responsible for directing the back-annotation transformation and uses the *Export Manager* service for retrieving the changes corresponding to the next step in the execution.

Business Process Execution Animation The execution derived from the counter-example can be presented on the graphical interface of the BPEL designer tool. Thus the results of the verification are illustrated with the same interface that was used to create the business process. The implementation extends the designer tool non-intrusively (i.e. without modifying the original implementation) with functions accessible from outside the tool. These functions provide support for setting the runtime state of the business process elements. The state is illustrated by colouring the graphical representation of the element (e.g. green for startable activity, red for erroneous variable).

The animation of the execution can be controlled with an easy-to-use interface either in a step-by-step or continuous way. It is possible to step forward and backward in the execution trace and the animation can be reset to the initial state. These functions correspond to the back-annotation transformation (see Sec. 3.4) Furthermore the animation can be toggled for fast stepping when several steps are executed at once.

Traceability visualization The traceability information generated by the transformations is used for multiple purposes throughout the approach. As these include use cases when the traceability model is handled manually (e.g. domain-specific requirement definition), the visualization of these models is essential for aiding the analysis. The model visualization component of the VIATRA framework was extended with support for domain-specific layouts[1] to visualize traceability models.

Summary In this section the overview of the integrated toolchain was described. First the existing and new tools and their connections are presented. The implemented graphical user interface is introduced which provides a frontend for the verification method and animation support for visualizing the results. Finally the traceability information can be visualized using a component of the transformation framework.

[1] Additional details of the visualization are found on the website: `http://home.mit.bme.hu/~ujhelyiz/pub/traceabilityvisualization.html`.

As for the integration of this work in the Sensoria chain, other modeling frontends such as Activity Diagrams in the UML4SOA notations can be easily integrated, however, this would need a modification of the editor to have the same simulation/back-annotation functionality.

4 Related Work

As this overview chapter presented several approaches of SENSORIA, here we cannot detail technical related work to all methods. These are covered in chapters of this book and other publications mentioned in Sec. 2.

It is worth mentioning that the above techniques were combined in several approaches, among others in [10] with a focus on management of non-functional properties in the development of Service Oriented Systems, or in [7] to achieve advanced composition analysis support. In [29], "patterns" for service engineering were collected.

4.1 Related model-driven analysis methods

Business Process Verification The verification of business processes has been thoroughly studied in the recent years. The early approaches only dealt with single processes, neglecting effects resulting from the fact that these workflows usually take part in multi party, distributed cooperations.

In [28] a subset of Petri nets was defined that models structurally sound workflows. Several structural properties of business processes could be analyzed. However, the proper utilization of a specific subset (omitting flow links) of the BPEL language can guarantee the soundness of the business process. In [14] an approach is introduced that enables model checking of business processes implemented in BPEL v1.1. The workflow implementation is transformed into Petri nets. The authors report that they have modeled the entire semantics of the language.

Foster et al. [6] propose Finite State Processes and the use of LTSA to verify Web services compositions. They use Message Sequence Charts to specify criteria, which is also one of the intended future research directions. The objective of [21] was to provide an analysis method that is capable of the modeling and verification of the four examples presented in the standard of BPEL v1.1 [3]. Hence event, fault and compensation handlers are not dealt with.

Garcia-Fanjul et al. in [8] describe a similar technique with a different purpose: they use SPIN to generate test cases for given requirements. However, their work aims at finding proper test suites for the implementation of BPEL processes and does not address design flaws. Recent works concerning the semantics of BPEL v1.1 processes, e.g. [19], are based on π-calculus. However, the definition of the requirements which can be checked by using this semantics is very general. To our best knowledge, [15] is one of the few works dealing with the data flow in service compositions.

Traceability Triple graph grammars [23] (TGG) is a technique where the correspondence between two different types of models is defined in a declaratively, this can be used to define synchronization model transformations. [4] uses TGG for UML model-based tool integration which supports the synchronization between various languages throughout the development process. [9] defines correspondence models interconnecting the source and target models of the incremental model synchronization also using TGGs.

QVT Relations [27] is an OMG standard with specific focus on bidirectional transformations for incremental model synchronization and defines a formalism similar to TGGs.

[22] includes similar traceability models to synchronize abstract and concrete syntax of domain-specific modeling languages. Live model transformations and synchronization requires precise traceability information although the use cases are often different from ours. The paper also includes a detailed evaluation of the state-of-the-art of traceability aspects.

[24] uses traceability to store the execution trace of the transformation which generates Alloy models from UML. The back-annotation transformation is automatically generated based on this trace using a QVT-based implementation. However traceability information is only used for automated execution, visualization is not supported.

5 Conclusions

This chapter presented an overview on the SENSORIA engineering approach where model-driven technologies can be connected to develop trustworthy service oriented systems. These technologies have some common problems (traceability, back-annotation, intuitive requirement definition, etc.) to face in order to be effective for a day-by-day use in engineering process. Such issues and an end-to-end solution were also presented (BPEL2SAL). Please note that this method is extendable, either the input can be an orchestration in UML4SOA (which would need additional transformations) or the analysis infrastructure can be replaced.

We envision that more and more "orchestrations" for service development would be composed in order to meet requirements of different service domains (e.g. automotive or financial systems) using the above techniques and exploiting the benefits of model-driven service engineering.

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

References

1. Eclipse BPEL Designer. `http://www.eclipse.org/bpel/`.
2. M. Alessandrini and D. Dost. SENSORIA Deliverable D8.3.a: Financ case study: Requirements modelling and analysis of selected scenarios. Technical report, S&N AG, August 2007.

3. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Version 1.1*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, May 2003.
4. S. M. Becker, T. Haase, and B. Westfechtel. Model-based a-posteriori integration of engineering tools for incremental development processes. *Software and Systems Modeling*, 4(2):123–140, May 2005.
5. E. A. Emerson. *Temporal and Modal Logic*, volume B, Formal Models and Semantics, pages 995–1072. Elsevier, 1990.
6. H. Foster. *A Rigorous Approach To Engineering Web Service Composition*. PhD thesis, Inperial College London, 2006.
7. H. Foster and P. Mayer. Leveraging integrated tools for model-based analysis of service compositions. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 72–77, Washington, DC, USA, 2008. IEEE Computer Society.
8. J. García-Fanjul, J. Tuya, and C. de la Riva. Generating Test Cases Specifications for Compositions of Web Services. In A. Bertolino and A. Polini, editors, *Proc. of WS-MaTe2006*, pages 83–94, Palermo, Sicily, ITALY, June 9th 2006.
9. H. Giese and R. Wagner. *Incremental Model Synchronization with Triple Graph Grammars*. Springer Berlin / Heidelberg, 2006.
10. S. Gilmore, L. Gönczy, N. Koch, P. Mayer, and D. Varró. Non-Functional Properties in the Model-Driven Development of Service-Oriented Systems. *Journal of Software and Systems Modeling*, 2010. Accepted.
11. L. Gönczy, Z. Déri, and D. Varró. Model Transformations for Performability Analysis of Service Configurations. pages 153–166, Berlin, Heidelberg, 2009. Springer-Verlag.
12. L. Gönczy and D. Varró. *Developing Effective Service Oriented Architectures: Concepts and Applications in Service Level Agreements, Quality of Service and Reliability*, chapter Engineering Service Oriented Applications with Reliability and Security Requirements. IGI Global, 2010. To be published.
13. C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro. : A calculus for service oriented computing. In *ICSOC*, pages 327–338, 2006.
14. S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In W. M. P. v. d. Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *in Proc. of BPM 2005*, volume 3649 of *LNCS*, pages 220–235, Nancy, France, Sept. 2005. Springer-Verlag.
15. R. Kazhamiakin and M. Pistore. Static Verification of Control and Data in Web Service Compositions. In *Proc. of ICWS '06*, pages 83–90, Washington, DC, USA, 2006. IEEE Comp. Soc.
16. M. Kovács, D. Varró, and L. Gönczy. Formal Analysis of BPEL Workflows with Compensation by Model Checking. *IJCSSE*, 23(5), November 2008.
17. H. A. López, F. Massacci, and N. Zannone. Goal-Equivalent Secure Business Process Re-engineering. In *Proceedings of the 2nd International Workshop on Business Oriented Aspects concerning Semantics and Methodologies in Service-oriented Computing (SeMSoC'07),*, 2008. TO appear as Springer Verlag book.
18. P. Mayer, A. Schroeder, and N. Koch. MDD4SOA: Model-Driven Service Orchestration. In *Proceedings of the 12th IEEE International EDOC Conference*, IEEE. IEEE, 2008.
19. M. Mazzara and R. Lucchi. A Pi-Calculus Based Semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, 2006.

20. C. Montangero, S. Reiff-Marganiec, and L. Semini. Logic-based detection of conflicts in APPEL policies. In Proc. of FSEN (IPM International Symposium on Fundamentals of Software Engineering), LNCS, 2007.
21. S. Nakajima. Model-Checking Behavioral Specification of BPEL Applications. *ENTCS*, 151(2):89–105, 2006.
22. I. Ráth, A. Ökrös, and D. Varró. Synchronization of abstract and concrete syntax in domain-specific modeling languages. *Journal of Software and Systems Modeling*, 2009.
23. A. Schürr. Specification of graph translators with triple graph grammars. In *WG '94: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163, London, UK, 1995. Springer-Verlag.
24. S. M. A. Shah, K. Anastasakis, and B. Bordbar. From uml to alloy and back again. In *MoDeVVa '09: Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–10, New York, NY, USA, 2009. ACM.
25. N. Shankar. Symbolic Analysis of Transition Systems. In Y. Gurevich, P. W. Kutter, M. Odersky, and L. Thiele, editors, *ASM 2000*, number 1912 in LNCS, pages 287–302, Monte Verità, Switzerland, 2000. Springer-Verlag.
26. M. Sorea. Bounded Model Checking for Timed Automata. *Electronic Notes in Theoretical Computer Science*, 68(5), 2002.
27. The Oject Management Group. Meta object facility (mof) 2.0 query/view/transformation (qvt), 2008. `http://www.omg.org/spec/QVT/`.
28. W. van der Aalst and K. van Hee. *Workflow Management Models, Methods, and Systems*. The MIT Press, 2002.
29. M. Wirsing, M. Hölzl, L. Acciai, F. Banti, A. Clark, R. D. Nicola, A. Fantechi, S. Gilmore, S. Gnesi, L. Gönczy, N. Koch, A. Lapadula, P. Mayer, F. Mazzanti, R. Pugliese, A. Schroeder, F. Tiezzi, M. Tribastone, and D. Varró. SENSORIA patterns: Augmenting service engineering with formal analysis, transformation and dynamicity. In *Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008)*, 2008. to appear.
30. K. Xu, Y. Liu, and C. Wu. Bpsl modeler – visual notation language for intuitive business property reasoning. *Electron. Notes Theor. Comput. Sci.*, 211, 2008.