Leveraging Integrated Tools for Model-Based Analysis of Service Compositions

Howard Foster* and Philip Mayer†
*London Software Systems, Imperial College London
London, United Kingdom
†Ludwig-Maximilians-Universität, Munich, Germany
Email: *hf1@doc.ic.ac.uk, †mayer@pst.ifi.lmu.de

***Abstract*—Developing service compositions, using multiple standards and implementation techniques, typically involves specifying service characteristics in different languages and tools. Examples are defining service composition behaviour, in the form of the Business Process Execution Language for Web Services (WS-BPEL) and a global service choreography policy, in the form of the Web Service Choreography Description Language (WS-CDL). Whilst there have been a number of model-based analysis tools reported, there is a lack of integration with development environments to support analysis of these different service artifacts. In this paper we present a short history of some of the analysis tools reported, discuss an appropriate criteria of accessible integrated development with analysis features and provide an example approach, called "Service Engineer" using our tools and integration work. The approach is supported by an integrated service tool-chain development environment known as the SENSORIA Development Environment. The aim is to provide an accessible, rigorous approach to analysing service compositions but with a simple, clearly defined interface in an integrated development environment.**

I. Introduction

As the rapid growth and use of system integration services (typically web services) continues, the work reported on the analysis of service behaviour models has also grown, with an application of formal methods proving valuable to assist engineers verify and validate service configurations prior to deployment. Most of this work has described formal modelling techniques with translations to process calculi or algebras. The verification and validation of these models is often not hidden from the engineer, which does not lay well with service engineers who are interested in the ease of the development and specification environment rather than the process and a non-contextual form of analysis results. Support tools for this verification typically require the user of the tools to *activate* a translation of service implementations, *feed* the translation through a model parser, *compile* the model, and instigate a *verify* option on the model checker. If the verification is undertaken once or twice on each implementation then this is not an issue, however, one can expect implementations to evolve over time (particularly as service reuse and maintenance increases) and thus, the ease of executing the verification steps frequently becomes equally as important as the result of the steps themselves.

In this paper we describe how our tool environment, integrated into the Eclipse Integrated Development Environment (IDE) provides and hides the type of functionality we discussed previously. Our aim is simple, to provide a rigorous verification and validation platform for service engineers, yet conceal as much as possible that is not relevant to the engineer but still provide highly accessible functions in as few steps as possible. We concentrate on support for two service composition specifications, namely the Web Service Business Execution Language (WS-BPEL) [1] and the Web Service Choreography Description Language (WS-CDL) [2]. Whilst there are others, we use these two to illustrate orchestration and choreography analysis for supporting service engineering.

The structure of the paper is as follows. In section 2 we describe a brief history of model-checking service composition tools, and discuss how they implement the verification steps mentioned previously. In section 3 we describe a framework for integrated analysis tools of service compositions. In section 4 we detail our approach and tool for service composition behaviour analysis and in section 5 we describe the integration of this tool into a development environment for analysis tools using the framework described in section 3. Section 6 concludes the paper with a review and comments on future work.

II. Background

There are a number of tools reported to support model verification and validation of service compositions. We believe we introduced one of the first tools LTSA-WS in [3], which presented an automated translation of service orchestrations described in the WS-BPEL and WS-CDL languages to the Finite State Process (FSP) [4] notation and subsequently allowed the user to compile and safety check models of service compositions. The tool was originally written as a plug-in to the Labelled Transition System Analyser (LTSA) also described in [4]. LTSA supports default safety (deadlock) and progress (property) analysis, but can also support Linear Temporal Logic (LTL) and Fluent property analysis. The tool framework is window and tabbed pane based and presented a simple WS-BPEL editor which required the user to switch between analysis editors, and invoke compilation, safety and animation manually. Another tool, called the Web Service Analysis Tool (WSAT) [5] is also a tool for the analysis and verifying of service compositions. They use an intermediate representation in the form of Guarded Finite State Automata

(GFSA). A translator converts WS-BPEL and WSDL input documents in to GFSA models and then synchronizability and realizability can be checked. Data expressions used within the service compositions (for temporary data and guarded actions) are abstracted to Promela code. The tool is command line based and requires the user to explicitly state the input processes and interfaces (WS-BPEL and WSDL documents) for analysis. WofBPEL [6] is also a tool for the analysis for service orchestrations in the form of WS-BPEL. The tool takes as input Petri-net descriptions of WS-BPEL, which are obtained as output from the BPEL2PNML tool accompanying the WofBPEL tool. The analysis, as with the others described in this background, considers the reachability of the composition processes but also provides a novel way to check for concurrent inbound messages and which ones need to be preserved in the process message queue at each step of a WS-BPEL process. The input to the tool is again command line driven and requires several steps to generate the required output and subsequent inputs between translation and model checker inputs. Tools4BPEL [7] also translates BPEL process descriptions to Petri net models using a BPEL2oWFM module. Again, safety analysis (specified using various properties) can be checked in the tool, but also there is an additional part of the tool, called 'Fiona' which checks for proper interaction models within the orchestrations and details operating guidelines for using each of the services appropriately. There has been some reports of direct editor integration with formal verification tools, such as with the ActiveBPEL tool in [8] which uses the UPPAAL model checker to check safety and timed analysis of orchestrations. The tool makes a good attempt to integrate with source editors yet does not provide the flexibility to work with other analysis tools.

Industry has also focused on providing some web service composition support, that is, above and beyond service orchestration editors and syntax validation. For example, the IBM Websphere Business Modeller [9] provides validation of process workflows, which in turn, can be exported to WS-BPEL compliant process descriptions. It is not known which techniques are used to perform this analysis, and it is believed that the analysis is limited to checking that they are complete flows (for example, that there are sufficient start and end nodes specified).

Our contribution builds on our earlier work and analysis tool suite by concentrating on a set of key design features to support enhanced development of web service compositions, and in particular the analysis of their specifications. We call the approach the 'Service Engineer Approach'. The approach has been formed as part of a European Union Global Computing 2 funded project called *Service Engineering for Service-Oriented Overlay Computers (SENSORIA)*. Further information on this project is available on the project website: *http://www.sensoria-ist.eu*.

III. Criteria for Tools

Our criteria consists of several steps to aid the engineer in performing analysis of service compositions. To begin with however, it is useful to define a framework for the approach based upon a number of principles. For service composition analysis tool support, we appropriately split the discussion into a series of criteria for evaluation taken from the work by Clarke and Wing in [10]. This criteria considers tool support from several viewpoints including; ease of learning, early payback, efficiency of developers time, increase in benefits, error detection, integrated development environment enabled, focus on analysis and support for evolutionary development. We now describe these and how we extend model based service engineering aspects of analysis.

Ease of Learning: In ease of learning the notations and tools should provide a starting point for writing specifications for developers who would not otherwise write them. It is not the intention of the service analysis tools to presume how the input for analysis is constructed. The analysis tools should provide a simple interface to allow user's to access the analysis functions with ease, and provide suitable results that can be understood without formal knowledge of the source specification translation model rules.

Early Payback: To provide methods and tools which give significant benefits almost as soon as user's begin to use them. Access to analysis functionality should be provided appropriately in the service composition development life cycle. For example, even at the stage of partial descriptions of service behaviour the user should be able to analyse incrementally the current specification and be presented with early results to guide the user to more an accurate, correct solution.

Efficiency of User's Time: Tools should make efficient use of a user's time, and in particular, turnaround time with an interactive tool should be comparable to that of normal compilation. Clarke and Wing also noted that developer's are generally more inclined to wait for tools that are known to perform more extensive analysis. Model checking analysis tools can require a large set of available resources and can carry out processes which take time to execute. Progress in analysis steps should be displayed where possible.

An Increase in Benefits: Benefits should increase as developers get more adept or put more effort in to writing specifications or using tools. The tool set should provide highly useful results, and presented in a context which is meaningful to the user. With service compositions this means providing the right type of analysis at the right point in the service development life cycle, with context related results displayed.

Integrated Use: The use of tools for formal methods should be integrated with that of tools for traditional software development. We have already mentioned that the analysis tool should not consider being a source editor for the service composition languages. Rather, it should complement the capturing of these specifications and provide an integrated, coherent approach to both undertaking the analysis and then using the results obtained from the analysis.

Error Detection: Methods and tools should be optimised for finding errors, not for certifying correctness. They should support generating counterexamples as a means of debugging. It should be assumed that correctness of the source documents

(i.e. that they are well-formed and follow the syntactical rules as defined in their specifications) has already been validated by editors. However, in the progress of translation to formal models and analysis, the tool should inform the user if a semantic value is invalid given the current source (above that of syntax). In translation we raise such violations as 'problems' and present these to the user during analysis. We also present examples of violations (such as the trace of actions specified that led to a violation) in an accessible form, in this case in Message Sequence Charts (MSC) notation.

Evolutionary Development: Methods and tools should support evolutionary system development by allowing partial specification and analysis of selected aspects of a system. As analysis is undertaken at points in the service life cycle, the user should be able to easily refine or elaborate on design or implementation and reiterate analysis to check that changes have the desired effect.

IV. The Service-Engineer Approach

Our initial approach and tool support for integrated analysis, takes a 2-dimensional view of service composition analysis, listed in table I. In a first dimension it considers core service composition artifacts being; orchestrations (service processes), interface (service descriptions), choreography (which could be argued are the main layers of service orientation) and resources (architecture dependent features of service compositions). From a second dimension it considers the analysis features of service compositions including; design, implementation, architecture configuration and deployment. Thus, one can use the approach to consider design for service orchestrations, or alternatively the deployment of collaborating processes and their architecture configuration in service choreography, or any aspect against the other in the matrix. We believe such an approach provides a much richer coverage of service composition development, accessible to engineers such that they can analyse compositions from different viewpoints (depending on the context of analysis). An overall integrated service behaviour analysis approach is suggested in Figure 1. The core of the approach is transforming some design or implementation artifact to relevant and detailed models for analysis (Model Generation). All inputs are provided from the perspective of a service engineer. We consider analysis of service orchestrations and choreography given input as design specifications (e.g. MSCs and UML2 diagrams), choreography policies (in the form of WS-CDL policies) and component descriptions (in the form of WSDL documents). Analysis in the approach provides features to compare each of these either as model validation (through animation) or verification through model property traces. Each feature considers behaviour analysis for a different element of service compositions.

Artifact	*Design*	*Impl.*	*Arch.*	*Deploy.*
Orchestration	MSC	WS-BPEL	Process	Engine
Interface	Function	WSDL	RPC	Host
Choreography	MSC	WS-CDL	Policy	Engine
Resources	UML**	Threads	Runtime	Server

TABLE I
Aspects of Analysis for Service Composition Behaviour

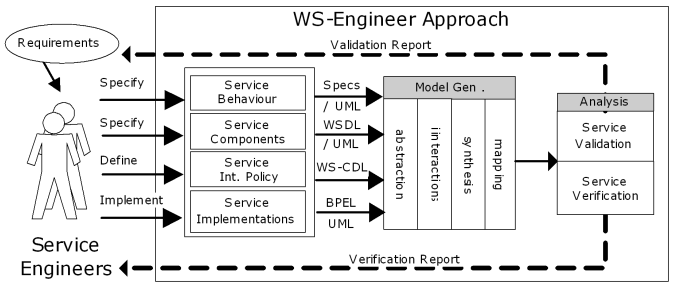

Fig. 1. WS-Engineer Approach to Service Composition Analysis

A. Analysis Features, Properties and Results

Service Design: Analysis of service composition design is achieved through the analysis of service specifications and in particular, the service behaviour specifications provided by the engineers. For example in our approach we provide analysis of MSCs describing the service composition specification. Infact, service design models can be used later with service implementations to formally model check that implementations fulfil their design specifications or vice-versa to check which behaviour sequences are supported by the design but not by the implementation. Here the engineer builds the MSC diagrams of service interactions and can check for implied scenarios (those scenarios which are not directly specified but emerge from the merging of scenarios). Such scenarios can be marked as 'positive' (accepted) or 'negative' (declined). They are then included or specifically omitted from design traces.

Service Implementations: Analysis of service composition implementations has focused on two areas. Firstly, the process of orchestration is analysed for safety (given varied properties of interest). Typically, this will be a check for deadlock freedom given the constraints imposed within the orchestration (such as linked sequences of operation or interactions specified in a concurrent process block). Secondly, orchestrations can be used as roles within a choreography (e.g. as part of a policy described in WS-CDL). Thus, the analysis of orchestrations can be checked with the property of a choreography role specification. Our work on service compatibility and obligations analysis [11] considered this area of analysis. Many of the approaches to orchestration analysis use WS-BPEL as a source input, and WSDL as the service interfaces. We described some of these tools earlier in section II. Our approach includes features to consider analysis of implementation against design, interaction in terms of collaborating orchestrations (linking interfaces and methods for service communication), and obligations between orchestrations and choreography.

Service Architecture and Deployment: We have described the analysis so far in terms of behaviour and interaction analysis. There is also a key issue of designing a suitable architecture configuration to host the service compositions (orchestrations, orchestration engine and server). Therefore, another consideration of analysis for service compositions is that of deployment. We specifically consider service design, implementation and architecture configuration in the analysis of deployment constraints. Constraints are imposed

by the configuration of resources, such as threadpools on the orchestration host server. To facilitate verification of service architectures and deployment environments our approach covered service composition analysis with resource constraints [12], which takes UML2 Deployment Diagrams (specifying orchestration processes, service hosts, servers and resource threadpools) and can check whether the behaviour of the processes collaborating on a particular configuration will lead to an exhausted resource usage. The result of this analysis is that design choices made in the architecture configuration can introduce deadlock situations when the processes are deployed for runtime.

B. Implementation in WS-Engineer

We have built an implementation of the analysis features mentioned in this section as a plug-in to Eclipse. The tool extends our previous work (as we described in section II on the tool LTSA-WS) by automating some of the tasks involved in analysis. The Eclipse plug-in is known as "WS-Engineer" and allows the user to verify designs, implementations and deployment models. Some features of the tool are fully automated, such as verifying orchestration processes (in WS-BPEL) against a design specification in MSC or WS-CDL specifications. WS-Engineer however, focuses largely only on behaviour. To cover different areas of analysis we were keen to extend the WS-Engineer environment as one of several tools and create an overall service analysis environment. The result was the creation of a analysis tool-chain approach, illustrated in Figure 2.

C. Tool-Chain Integrated Analysis

In this section we have described some common analysis features with reference to the behaviour of service compositions. Clearly, such analysis features may be required to be undertaken either alone or as a sequence of actions. We have developed an integration platform, known as the *SENSORIA Development Environment (SDE)* which acts as host for different analysis tools. These different types of analysis tools effectively add a third dimension to the features listed in table I. Thus, where we have described the "Service-Engineer" approach in terms of behaviour, there is also timing, performance and other types of analysis which can be carried out on service compositions. We believe that the SDE (fully described in section V) will highly benefit the service engineer in carrying out analysis features such as those described in this section.

V. The SENSORIA Development Environment

The SENSORIA Development Environment (SDE) [13] is a service modelling, development, and analysis platform built into the industry-standard Eclipse IDE [14]. The aim of the SDE is to provide the various tools required for developing services in one consistent and integrated environment, offering state-of-the-art research techniques in an easy-to-use fashion to developers. Advantages of the SDE over other tool integration platforms include: *An SOA Architecture* - The SDE itself

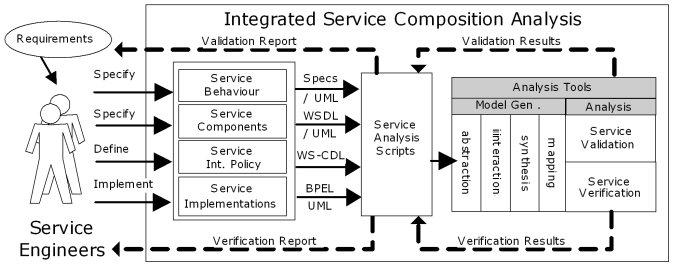

Fig. 2. Integrated Approach to Service Composition Analysis

is based on a Service-Oriented Architecture, allowing easy integration of tools and querying the platform for available functionality. The analysis tools hosted in the SDE are presented as discoverable, technology independent services. *A Focus On Usability* - Sophisticated analysis tools are often based on mathematical models and formal languages. To allow developers to use such tools without requiring them to understand the underlying formal semantics, the SDE employs automated model transformations which translate between high-level models and formal specifications, thus closing the gap between those two worlds. *A Composition Infrastructure* - As development of services is a highly individual process and may required several steps and iterations, the SDE offers a composition infrastructure which allows developers to define commonly used workflows as an orchestration of other tools.

In the next sections, we will detail the SDE platform and how it addresses the criteria outlined in section II. The SDE and plug-ins, including the WS-Engineer tool are also available for download at: *http://www.sensoria-ist.eu*.

A. Framework

The SDE is built on the Eclipse platform and its underlying service-oriented OSGi framework [15]. It contains three core features by which it extends the Eclipse platform, thus creating a generic modelling, development and analysis environment:

Firstly, the SDE contains a registration and discovery service in the spirit of a UDDI registry. Upon installation, a tool is registered with its offered functionality in the form of public API functions. The functions offered by each tool are made available for querying and execution by the SDE core. Secondly, the SDE allows scripting of all available functions, which enables developers to create new tools with new public API functions to be used like any other tool (i.e. the scripts become services themselves). In this fashion, tools inside the SDE may be combined on various levels with different complexity. Scripts may be available from tool authors for bridging the gaps between their tool and others, or may be written from scratch by the users themselves. Figure 3 illustrates a script between the WS-Engineer tool and PEPA tool for safety and performance analysis of a WS-BPEL orchestration. Finally, the SDE contains a generic infrastructure for working with tool APIs. Through various integrated tools, the SDE currently offers functionality which falls into three major categories:

Modeling Functionality: This includes graphical editors for familiar modelling languages such as UML, as supported by industry-standard tools such as the Rational Software

```
function checkBPEL(bpel) {
        // transform BPEL to SRMC (input to srmc tool)
        bpel2srmc = sCore.findToolById("bpel2srmc");
        srmcCode = bpel2srmc.transform(bpel);
        // perform analysis with PEPA/SRMC tool
        srmc = sCore.findToolById("srmc");
        markovChain = srmc.getMarkovChain(srmcCode);
        distribution = srmc.getSteadyState(markovChain);
        throughput = srmc.getThroughput(markovChain);
        // back annotation
        bpel2srmc.reflect(bpel, distribution, throughout);

        // transform to LTSA FSP (input to WS-Engineer)
        bpel2ltsa = sCore.findToolById("wsengineer");
        ltsaCode = bpel2ltsa.bpel2fsp(bpel);
        // perform analysis with WS-Engineer
        wse = sCore.findToolById("wsengineer");
        result = wse.analyse(ltsaCode);

        trace = null;
        if (result.hasErrors()) {
                trace = ltsa.mscFromLTSATrace(result.trace);
        }
        // return results
        results = [bpel, result, trace]
        return results;
}
```

Fig. 3. JavaScript of two Analysis Tool Features (safety and performance) in the SDE which can then be installed as a new Analysis Service

Architect, which allow for intuitive modelling on a high level of abstraction. However, there are also text- and tree-based editors for use by those users who have more rarefied tastes and would prefer to express their models in a process calculus instead of UML.

Model Transformation Functionality: The SDE offers automated model transformation from UML to Web Service standards to process calculi (and back) to bridge the gap between these worlds, thereby enabling users to stay on a modelling level while still enjoying formal analysis methods.

Formal Analysis Functionality: Finally, the SDE offers model checking and numerical solvers for stochastic methods based on process calculi code defined by the user or generated by model transformation.

Figure 4 shows an example of the SDE used for checking the safety of a BPEL process. On the left hand side, the SDE tool browser is shown which allows developers to access the tool registry (Modeling Functions). In the top middle, information about the tools discussed in section IV are shown (Transformation and Analysis). Below the tool functions are the results of analysis (in this case a graphical trace of a violation in the BPEL process). The functionality available through this API has been used for generating the output in the lower parts with the BPEL process shown above as input. The script, such as that described previously, can enact these steps automatically.

B. Evaluation with Criteria

As an integrated platform for service development, the SDE offers more than the sum of its tools: Through composition, the combined tools offer more streamlined and easy-to-use functionality, which is reflected in the evaluation of the criteria discussed in section II.

Ease of Learning: Through automated model transformations, the SDE enables developers to stay within their modelling language of choice and still enjoy formal analysis support. The SDE contains tools for working with UML models, converting these models to Web Service standards like BPEL and WSDL or to process calculi. Through the scripting interface, the model transformations and formal analysis can be hidden; only the results from the analysis need to be shown.

Early Payback and Support for Evolutionary Development: The tools available within the SDE are able to deal with various kinds of input, for example high-level UML models (which are available early in the development phase) or specifications in Web Service standards (which might be a starting point for re-engineering efforts and maintenance). Thus, analysis functionality is available at each step in the development process, including the very first ones.

Efficiency of Users Time: Through the ability for creating tool orchestrations, the SDE offers to repeat several steps in an analysis in an automated fashion, thus removing the need for the user to manually click through a complicated and tedious process themselves. Also, some of the tools offered within the SDE perform their analysis on a server instead of the client, which enables them to process models more swiftly and asynchronously, thus adding to the overall efficiency of development.

An Increase in Benefits: The SDE contains tools of various levels of sophistication. Many offer both simple verifications for checking basic functionality, but also complex analyses which require more information from the developer in order to be carried out. As the SDE is an open platform, tools may be installed and used as appropriate, enabling the developer to simply plug-in more complex tools as they grow more experienced. Finally, through the orchestration mechanism, more complex analyses can be included by means of simply extending a script.

Error Detection: The SDE propagates error contexts through tool API invocations and workflow. Through the integrated model transformations, verification results can be translated back to the input language of the user (e.g. UML), thus allowing a single context of representation in tool use.

Integrated Use: The core function of the SDE framework is to integrate both development environment and tool usage. Available tools in the SDE share common models and transformations features to provide consistent source input and output. These tools are chained together to present a single interface. In addition, as the SDE is based on the Eclipse platform other plug-ins available for this platform can be utilised.

VI. Conclusions

In this paper we have presented a brief background of analysis tools for service compositions, and have discussed a criteria for leveraging tools within integrated model-based analysis environments. We also presented 'Service-Engineer', an approach which considers the main areas of service compositions and the analysis of different stages of development. We believe our main contribution is to provide an integrated analysis tool-chain development environment, and we described the SENSORIA Development Environment in

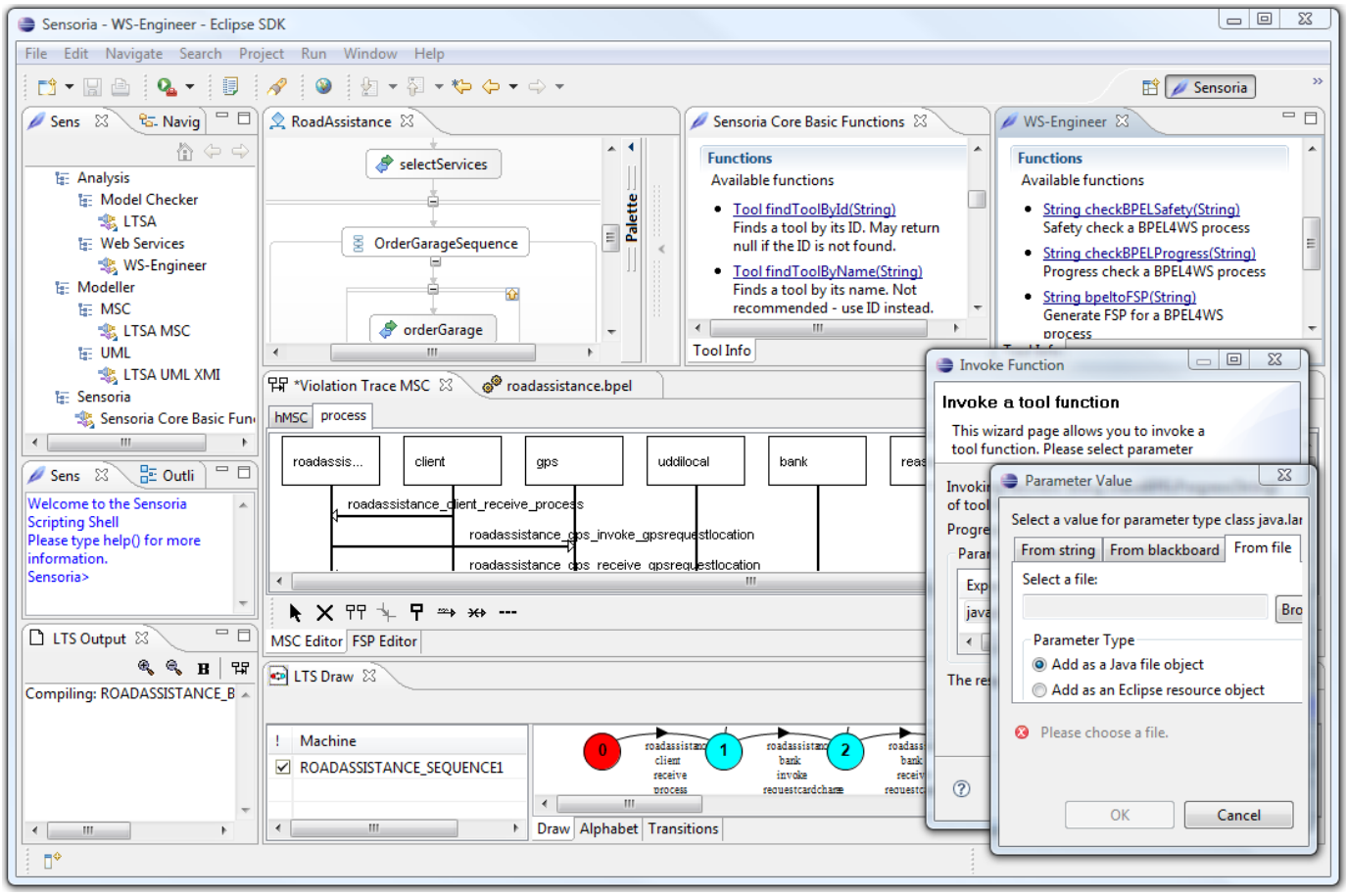

Fig. 4. The Eclipse, SENSORIA Development Environment and Integrated Analysis Tools (such as WS-Engineer)

section V which provides a tool framework to orchestrate (as scripts) analysis of service compositions using registered tool services. Our future work will explore building these scripts to perform composed analysis sets given different contexts of service compositions. For example, a script may be executed to verify the safety of service orchestration processes against resource constraints, but also provide estimates of service performance (e.g. response time) given the same architecture configuration. Evolutionary development allows the engineer to iteratively change parts of the service compositions repeating the analysis scripts and gaining a much richer insight in to overall effectiveness of the design and implementation. The authors are supported by the EU FET-IST Global Computing 2 project SENSORIA (IST-3-016004-IP-09). The original LTSA tool was developed by Jeff Magee and Jeff Kramer of the Department of Computing, Imperial College, London.

REFERENCES

[1] Alexandre Alves et al., "Web service business execution language (ws-bpel) v2.0", Oasis standard, OASIS, 2007.

[2] Nickolas Kavantzas et al., "The web service choreography description language v1.0", W3c recommendation, W3C, Nov. 2005.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions", in *Proc. of the* 18^{th} *IEEE Int. Conference on Automated Software Engineering*. 2003, pp. 152–161, IEEE CS Press.

[4] J. Magee and J. Kramer, *Concurrency - State Models and Java Programs - 2nd Edition*, John Wiley, 2006.

[5] Xiang Fu, Tevfik Bultan, and Jianswen Su, "Wsat: A tool for formal analysis of web services", in *16th International Conference on Computer Aided Verification (CAV)*, Boston, MA, 2004.

[6] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede, "Wofbpel: A tool for automated analysis of bpel processes", in *ICSOC*, 2005, pp. 484–489.

[7] Niels Lohmann, "A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN", Informatik-Berichte 212, Humboldt-Universität zu Berlin, Aug. 2007.

[8] Yi Qian, Yuming Xu, Zheng Wang, Geguang Pu, Huibiao Zhu, and Chao Cai, "Tool support for bpel verification in activebpel engine", in *The Australian Software Engineering Conference (ASWEC)*, Melbourne, Australia, April 2007.

[9] IBM Corp, *Best Practices for Using Websphere Business Modeller and Monitor (REDP-4159-00)*, IBM RedBooks, 2006.

[10] Jeannette Marie Wing Edmund Clarke, *Formal methods : state of the art and future directions*, Pittsburgh, Pa. : School of Computer Science, Carnegie Mellon University, 1996.

[11] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer, "Compatibility for web service choreography", in *3rd IEEE International Conference on Web Services (ICWS)*, San Diego, CA, 2004a, IEEE.

[12] Howard Foster, Wolfgang Emmerich, Jeff Magee, Jeff Kramer, David S. Rosenblum, and Sebastian Uchitel, "Model Checking Service Compositions under Resource Constraints", in *the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, 2007.

[13] Philip Mayer and Hubert Baumeister, "Report on the sensoria case tool (d7.4b)", Deliverable, EU Project Sensoria, Aug. 2007.

[14] Eclipse.org, "The eclipse open development platform v3.3", 2008, Available from: http://www.eclipse.org.

[15] OSGi Alliance, "Osgi specification rel.4", 2007, Available from: http://www2.osgi.org/Specifications/.