

# Tool Support for Safety Analysis of Service Composition and Deployment Models

Howard Foster

*Distributed Software Engineering, Department of Computing,  
Imperial College London, 180 Queen's Gate,  
London SW7 2BZ, UK  
howard.foster@imperial.ac.uk*

## Abstract

*This paper reports on an implementation for tool support of model-checking collaborating service compositions with deployment configurations under resource constraints. The implementation accepts UML Deployment Diagrams with an applied service deployment profile and one or more WS-BPEL orchestrations that are assigned to web servlets and servers in this deployment. Using model-checking techniques the tool can determine whether the configuration of deadlock-free service orchestration processes introduce deadlock scenarios when combined with resource constraints of a deployment environment. The implementation is built upon a tool suite, called WS-Engineer, which is aimed at assisting service engineers in constructing and testing various aspects of a service engineering approach, including orchestration, choreography and deployment artifacts. The tool integrates as a plug-in for Eclipse, alongside the IBM Rational Software Architect tool suite and others. A case study based upon a complex service grid solution, for analyzing chemical markup patterns, is used to demonstrate the accessible and practical nature of the solution.*

## 1. Introduction

Recently, the main focus of service orchestrations and behaviour analysis has been on model-checking service behaviour for safety properties, where safety is concerned with checking deadlock freedom in single or multiple service orchestration processes. We were particularly interested in extending the support for analysis in to the service architecture domain and in [1] an approach to model-checking service orchestrations under resource constraints with the extendable Architecture Description Language (xADL2) for specifying deployment architecture configurations was reported. The approach highlighted the need to consider deployment configurations in addition to the behaviour models so as to provide greater assurance that service orchestrations would be dead-lock free at runtime. Additionally, since this work was reported a

number of service profiles have emerged to allow service engineers and architects to model service configurations from the perspective of service behaviour and service deployment. Our extended approach now aligns with these profiles and we can demonstrate an integrated tool-chain approach to service engineering using mainstream tools, such as Rational Software Architect [2] from International Business Machines (IBM).

This paper details the implementation of the approach using the Unified Modelling Language (UML) Version 2 [3] and WS-BPEL 2.0 [4] specifications. The contribution is aimed at providing a highly accessible and practical tool suite to assist service engineers in detecting and refining service architecture configurations prior to deployment and running the service solution. The implementation is supported by a case study provided by University College London (UCL) in their solution for a complex services grid implementation for analyzing chemical markups for deficiencies. In section 2 a background to model-checking service compositions and deployment architecture analysis is discussed. In section 3 the approach and tool suite design to support this is detailed. In section 4 the requirements of a deployment configuration profile for UML and service compositions are defined, and in section 5 the transformation of orchestrations and deployment configurations to models are specified. In section 6 the analysis of these models and an iterative approach to refining solutions based upon the results of this analysis is presented. Section 7 concludes this paper with consideration of the results and future work in this area.

## 2. Background

A Web Services Architecture (WS-A) is a set of conceptual elements which define a common set of standards between interoperating components, running on different platforms and/or frameworks. The W3C definition of WS-A [5] includes a note that there is no restriction on how these services are implemented or combined to provide more complex compositions. The web services standards stack (including specifications for data, interface, service description, orchestration and choreography) is evolving to support various aspects of

creating web services architectures, yet there remains an ambitious task of building systems on such architectures and it is closely aligned with the capabilities of technology support. Amongst these standards is the Business Process Execution Language for Web Services (WS-BPEL) [4], for service composition and orchestration. BPEL has gained significant support by both industry and academia. The notation in this specification aims to provide a standard description of processes which interact with a number of service partners, and therefore research has explored modelling these orchestrations to provide both design and implementation support in building complex processes.

The deployment of web services into a web service container means that the resource demands of a web service during an invocation are controlled by a container in cooperation with the underlying operating system. Containers typically terminate if they exhaust memory or if the number of file descriptors that can be open at any one time is exceeded. The behaviour for the management of connections and threads is different though and a container would have a fixed and configurable pool of these resources. Web service compositions using BPEL are executed by a specialist container, sometimes called a BPEL engine or a BPEL run-time environment. These containers will again use resources. BPEL engines will, for example upon receiving a SOAP message to start a BPEL process, instantiate this process and execute it in a separate thread concurrently with other ongoing BPEL processes. Both Web service and BPEL containers typically map these threads efficiently to a set of operating system threads. The amount of operating system threads however, is finite due to the finite amount of memory required to handle the stack segment of the thread. Administrators must therefore carefully configure the thread pools to avoid exhaustion of the operating system resources.

## 2.1 Related Work

The analysis of web service orchestrations has included mapping BPEL to Finite State Processes [1] to discover deadlocks and progress violations. Similarly in [6] Petri Nets are used for control logic checking of BPEL and specifically describes analysis for isolating “redundant messages” that are not necessary if a certain activity has been performed. Alternatively, [7] uses Petri net-based models to represent web service composition flows independently of a particular specification by creating a “web service algebra” (in BNF-like notation). There is however, little coverage of how this maps to current standard web service composition languages (such as BPEL4WS or WS-CDL). In [8] web service compositions are described in the Language of Temporal Ordering Specifications (LOTOS). The authors extend a mapping between the algebra and BPEL4WS by providing rules for

a partial two-way transformation. Fu [9] reports an analysis tool based upon translation of BPEL descriptions to Promela and analysed using the SPIN tool. They apply limited data expressions for state variable access analysis.

A missing part of all these approaches to analysis are the constraints posed on an orchestration execution given a service deployment architecture configuration. There have been few publications on the design of web services architectures, behaviour of compositions and resource management. Closely related to these topics however, are the works published on component resource management and evaluation of their resource usage. In [10] the authors present formalism for specifying component interfaces that expose requirements on limited resources. Their formalism permits an algorithmic check if a composition of components exceeds the available resources (e.g. network buffer overflows) and suggests an optimal configuration. In [11] the authors model the states of a set of printer components which uses memory resources for pages, fonts etc. Using probabilistic calculus their approach also permits compositional resource usage reasoning. Alternatively, [12] uses Labelled Transition Systems (LTSs) in a theoretical framework capable of conducting analyses of discrete-event processes that interact through shared, discrete resources. The approach defines a simulation language (called DEMO), which mimics the acquisition and release of available resources in both synchronous and asynchronous processes.

Our extended approach in [1] detailed resource analysis with service compositions and here we present an implementation of the approach by building resource models and their management in UML and checking resource compositional safety with BPEL compositions.

## 2.2 Chemical Grid Services: A Case Study

The problem of services and resources arose in a sizable theoretical chemistry work-flow (called Molpak) using BPEL. The workflow, developed at University College London, is used to predict organic crystal structures from a chemical diagram. In order to structure the complex workflow, it was decomposed it into a number of component workflows, with each exposing a web services interface. A client component invokes a root workflow, which then invokes up to 38 InvokeMolpak workflows (to generate the chemical mark-ups) in parallel. These, in turn, invoke a service workflow in order to submit jobs to a grid resource manager. The service workflow uses an analysis service [13] to submit a job and then polls a job monitoring service until the results are returned. Once these jobs are completed a further 200 parallel service workflows are instantiated that then in turn each again submit a job to the service workflow, causing the same process of job submission and polling for results. The workflow and its orchestration in BPEL are defined in

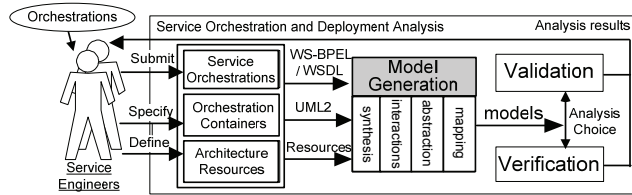
detail in [14]. In the following sections we describe how we modelled the architecture and orchestrations and provided analysis to determine how the behaviour and deployment configuration specified caused a problem of exhausted resources.

### 3. Approach and Tool Design

Our approach aims at supporting service administrators (architects, engineers and system operating staff) in configuring appropriate deployment architectures for the provision of operating service orchestrations on service servers. As a high-level goal the tool should represent the steps involved in the approach and provide a mechanical, automated analysis, where it is possible.

#### 3.1 The Approach

A summary of the approach is illustrated in Figure 1. The first step in the analysis process is to gather a set of one or more service orchestrations (in WS-BPEL format), one or more deployment architecture configurations (in UML2) and a set of resources – either included in the deployment configuration or specified separately.



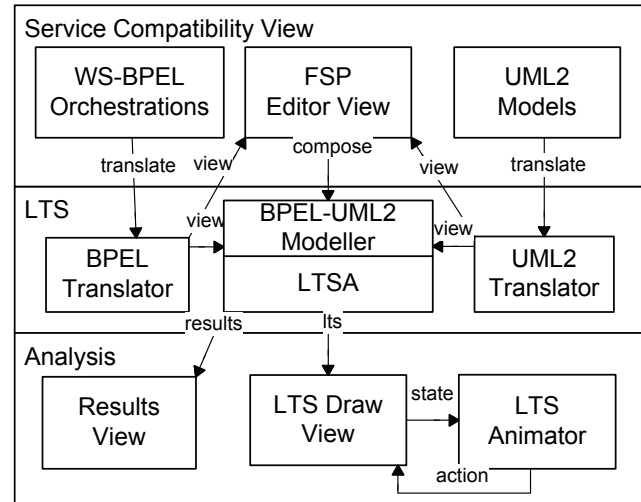
**Figure 1 Approach to Analysis of Service Orchestrations and Deployment Configurations**

The second step analyses these inputs and produces a set of models (through model synthesis and abstraction of interactions and their mappings) based upon the behaviour of the supplied orchestrations. Orchestration models are produced as a series of process compositions and resource models are produced based upon the type and usage in the deployment specification. The third step involves checking the process models against resource models based upon web server and servlet resource usage profiles (discussed later). The analysis also provides validation routines to allow the engineers to walk-through orchestration and deployment usage to assess if specifications have detailed the appropriate actions. The reader is invited to refer to the work in [1] for more details on the approach, analysis and verification techniques.

#### 3.2 The Tool

We have taken the approach and built a tool suite around these steps. The tool design is based upon the existing

WS-Engineer tool suite [15] providing analysis of service orchestrations, choreography and collaboration.



**Figure 2 LTSA WS-Engineer Extensions for Service Orchestration and Deployment Analysis**

WS-Engineer is built on top of the Labelled Transition System (LTSA) [16] which takes as input a model described in the Finite State Processes (FSP) notation. FSP is based upon the Communicating Sequential Processes (CSP) algebra but is formed to be an easily machine readable process algebra to formally describe processes and has been extended to include fluent (liveness), stochastic and probabilistic property analysis of process models. An extended tool design introduces a number of new components to facilitate the steps mentioned previously, including translation of UML models to processes and the modelling of UML deployment resources with the behaviour specified in the service orchestrations. The extended tool design architecture is illustrated in Figure 2.

The process of analysis in the tool is as follows. A “Service Compatibility View” provides a single interface for the engineer to specify the appropriate inputs to the process. The engineer can then select to verify that the models of behaviour and deployment configuration are “safe”. This action firstly invokes the translators for both WS-BPEL orchestrations and UML2 Deployment configurations. The result is two models, one representing the parallel composition of WS-BPEL orchestrations (with connectors between orchestrations where they interact) and a set of resource pools, one for each resource of a web servlet (configured on a web server) and specified as a resource type. The models are passed to a “BPEL-UML2” modeler, which analyses the models and defines appropriate properties to safety-check for the exhaustion of resources given the nature of the behaviour described in the orchestrations. To provide detail on the

implementation of these functions, we consider both deployment configuration specifications and orchestration modelling in the following sections.

#### 4. Deployment Specifications

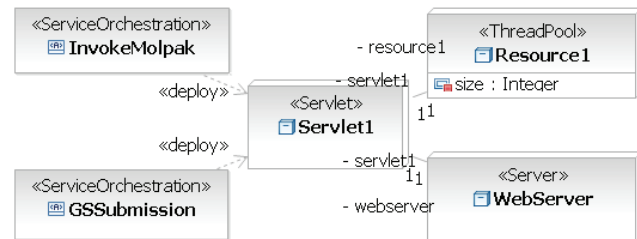
UML2 provides a Deployment (configuration) Diagram notation to support describing a static view of the run-time configuration of processing nodes and the components that run on those nodes. This is typically used to represent both hardware nodes, software artifacts (that are installed on the hardware nodes) and middleware components which connect the disparate nodes to one another. Nodes may contain other nodes, or be associated by a relationship connector. A special type of association in UML2 is the “deploy” relationship. This connects an artifact or node with another, and specifies that it is deployed on that given node. As with all UML2 notations, stereotypes provide a level of semantic identification on the models. This enables the user of these specifications to identify the appropriate type of node or artifact for deployment. UML2 by default provides a standard set of deployment stereotypes, such as <<device>>, <<executable>> etc. The approach discussed in section 3.1 requires an extended deployment profile to include service orchestration and service server elements. The profile extension is detailed in Table 1. The AN column represents ‘Artifact’ (A) or ‘Node’ (N) respectively, and stereotype constraints represent the attributes and relationships that must exist for a valid element of that type in a deployment configuration.

**Table 1 Service Deployment (Extended) Profile**

Profile Stereotype	A	UML2 Base	Stereotype Constraints	Semantic Representation
Service Orchestration	A	<i>Executable</i>	<<deploy>> assoc. Start	A service implementation (e.g. BPEL)
Servlet	N	Base Node	1.<<deploy>> assoc. end 2.Resource Map	Orchestration Engine Servlet
ThreadPool	N	Base Node	Attribute: ‘size’ (integer)	ThreadPool Resource and Allocation
Server	N	<i>Device</i>	n/a	Servlet Engine and Web Host.

Firstly, <<ServiceOrchestration>> artifacts stereotype the BPEL service orchestrations are linked with <<Servlet>> nodes which represent the container for deployed service orchestrations. The relationship between orchestration and servlet is provided by connecting an orchestration and servlet with a <<deploy>> stereotype association (provided in the standard deployment profile). Additionally, a resource map is required to specify how the servlet allocates and releases resources (for the

orchestration). We discuss this map further in section 5. A <<ThreadPool>> node is also associated with a servlet. The ThreadPool has an attribute “size” which represents, this case, an Integer amount for a resource pool constraint. Lastly, a <<Server>> node represents an actual web server host for one or more <<Servlet>> nodes. An example deployment configuration for the Molpak case study described in section 2.1, and using these various profile elements, is illustrated in Figure 3. Initially, a solution for the case study service orchestration was deployed to a single web servlet (represented in Figure 3 with two orchestrations deployed to Servlet1).



**Figure 3 UML2 Deployment Diagram with Service Deployment Profile Nodes and Artifacts**

UML2 Deployment specifications can be created in Rational Software Architect (RSA) tool by International Business Machines (IBM). RSA supports exporting any UML model, with profiles, to the OMG XMI XML standard. We use these exported models as source input in to the BPEL-UML2 modeller. We now examine how the models of the service orchestrations and deployment specifications are generated in our tool suit to generate analysis models for the behaviour and resource constraints specified in these models.

#### 5. Models of Service Behaviour and Resources

A service behaviour model, namely the two key orchestrations of the Molpak case study described earlier, is generated by a BPEL translator module in WS-Engineer. We have reported on the basic mechanism to translate BPEL to FSP in our earlier work in [15]. The translator is extended to support orchestration choreography mapping (to link actions between orchestrations), and we also provide a BPEL-UML2 Deployment modeller to support generating the models of mapping resource usage actions and to actions specified in the BPEL. The process of translation begins by generating a composed process for each of the orchestrations. In WS-Engineer a BPEL XML document is translated using a simple “loadBPEL→translateToFSP” pair of calls. For each orchestration, additional choreography mapping is constructed using port connectors. A port connector

represents the sequence of actions for communication between processes. In this case the web service model is “request→receive” (to model an invocation and receive action between partners) followed by “reply→receive” (to model sending the reply and receiving the reply back in the calling partner service). The result of composing the process models with port connector models is a sequential process of combined behaviour between orchestrations. In the case study we therefore had a link between InvokeMolpak and GSSubmission services, such that the invocation of the submission job in the molpak workflow was interleaved with the receive action of the submission job service. The processes are linked using the Web Service Description Language (WSDL) interfaces supplied with the BPEL processes. The full process for this is described in [15]. A final step in translation appends a create instance and terminate instance to represent when the service process are created and destroyed on the deployment architecture. The result of calling “toFSP” provides a source model document representing the parallel composition of all actions in all the services supplied as input. The composition in FSP can be compiled to Labelled Transition Systems (LTS) using the LTSA module included in WS-Engineer. Example models produced for the Molpak service orchestrations are illustrated in Figure 5.

The second stage of modelling involves translating the UML2 Deployment Diagrams to resource models. As input, the WS-Engineer translates a UML model and identifies the Nodes and Artifacts specified using the Service Deployment profile (discussed in section 4). The process is as follows. Firstly, an internal model of the Nodes and Artifacts is generated directly from the model source. The BPEL-UML2 modeller then generates a standard resource pool model (TPOOL) using an FSP model to represent “get” and “put” actions (to describe the allocation and release of threads from the pool). A threadpool model is illustrated in Figure 4. The BPEL-UML2 modeller then identifies all the UML Nodes that are of type ‘Servlet’ and identifies it’s associations with other Nodes that are of type ‘Threadpool’. For each threadpool of a servlet, a thread resource pool is created by creating an instance of the TPOOL process, uniquely identified by

a sequence number. Thus for the deployment specification illustrated previously in Figure 3, a single resource pool is created for the servlet ‘Servlet1’ with a threadpool of ‘size=10’.

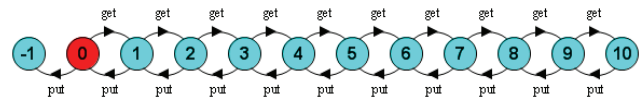


Figure 4 Threadpool (Behaviour) Model as LTS

To represent the allocation and releasing of thread resources with that of the behaviour specified in the service orchestrations deployed and using the thread pool we map the actions of each service orchestration to either a “get” or a “put”. The mapping semantics for this are attributed to the orchestration engine, and are associated with the ‘Servlet’ Node in the Deployment specification (see section 4). The map documents which service actions acquire and release a thread. In the Molpak case study the technology architecture used was based upon the Apache Tomcat 5.5 Servlet/JSP Container [17] and the ActiveBPEL Engine [18] which runs on the J2EE environment. In the case of ActiveBPEL we have found that the architecture described in the ActiveBPEL Engine documentation required further analysis of the engine logs to determine when threads are acquired or released. Given this knowledge, we can mechanically scan all the actions of the service orchestrations and map them to either a “get” or “put” action appropriately. The BPEL-UML2 modeller performs this task by abstracting the map (in XML) from the Servlet association and then traversing through each orchestration action, identifying its mapping by the operation (invoke, reply, receive etc) and appending the mapping to a pool remapping model. A similar process is undertaken if the deployment model contains multiple servlets and/or threadpools associated with the servlets. However, when the actions are mapped to multiple threadpools, only those orchestration actions associated with a threadpool are mapped to the particular threadpool resource actions. The resulting models can now be used for analysis of safety for the behaviour of the services and resources defined.

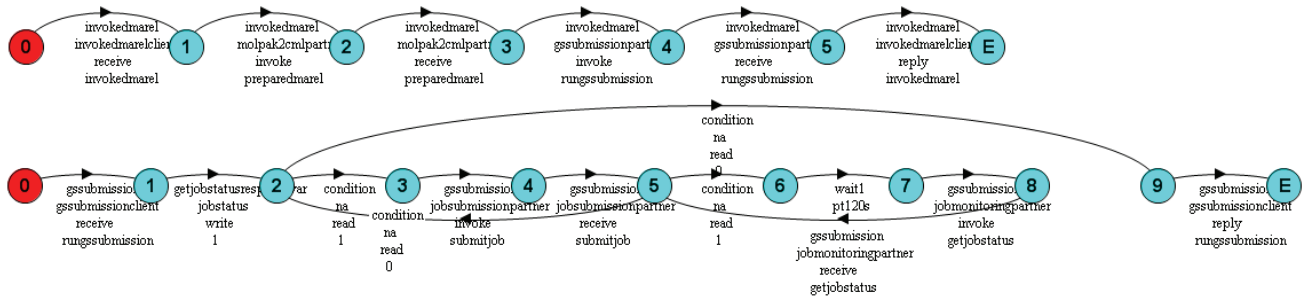


Figure 5 BPEL Service Orchestration (Behaviour) Models as LTS

## 6. Analysis and Evaluation of Tool Suite

### 6.1. Analysis and Results

The model (in FSP) generated by the translation described previously can be parsed, compiled and checked for “safety” using the LTSA module included within the WS-Engineer tool suite. A safety-check performs a search of all possible transitions and reports on any dead-locks within the model. In the case of our case study example, a resulting dead-lock violation was reported for service orchestrations and resource pools all located on one servlet and server. The WS-Engineer tool can map the trace results of violations and present back to the user in the form of Message Sequence Charts (MSCs). The violation from the case study is illustrated in Figure 6.

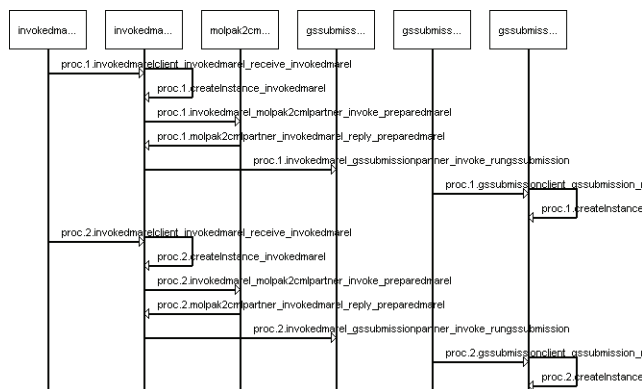


Figure 6 MSC of Safety Violation Trace

The reason for this deadlock is an exhausted resource thread pool allocation, whereby a request to create a new instance of the GSSUBMISSION orchestration process is never fulfilled due to the interaction causing an activity of requesting for a new thread (for which there are none to provide). Correcting this violation could be achieved in several ways. Firstly, we could simply increase the size of the threadpool. However, re-analysing this deployment configuration provides a similar violation. The reason is that the GSSUBMISSION orchestration waits and polls for the reply of the analysis web service (for defect analysis of chemical mark-up specifications). As both orchestrations share the same threadpool and servlet, this exhibits a potential violation whenever an additional client request is made to the InvokeMolpak orchestration (and instance created). Addressing this issue, the case study architects referred back to the deployment configuration and added a second servlet and threadpool configuration. The service orchestrations were then split across two servlets and the shared resource issue was resolved (resulting in no trace to violation in the model). The second attempt for deployment configuration is illustrated in Figure 7.

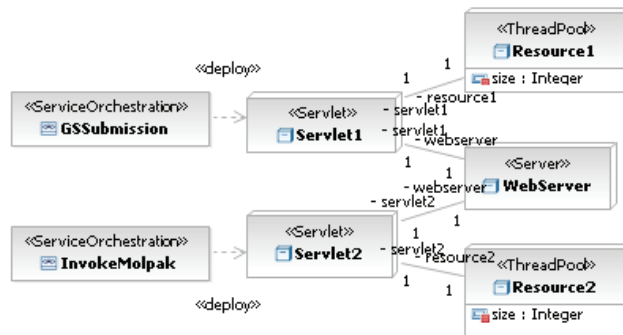


Figure 7 Refined UML2 Deployment Diagram for Service Orchestration and Resources

### 6.2. Tool Integration

The WS-Engineer tool, including the BPEL-UML2 modules described in this paper, are now available as an integrated plug-in for the Eclipse development environment. The migration to Eclipse has proved useful in collaborating with other plug-in projects in the form of editors and we feel the potential for our work can be leveraged by others in the wider Web Services Architecture development goals. In this work, we have integrated WS-Engineer in to the Rational Software Architect (RSA) tool suite produced by IBM. Service Engineers and Architects can use a single integrated development environment (IDE) to design, implement and test service orchestrations with deployment specifications for safety prior to any deployment actually being undertaken. A view of the tools integrated together is illustrated in Figure 8.

### 6.2. Ease of Learning

Our approach and the tool built to support it, aims at providing the following criteria for its ease of learning and carrying out verification and validation. Firstly, the design specifications are based on the OMG UML standard which is supported by various methodologies and toolsets. The use of UML is widely undertaken and understood in industry. Additionally, the service orchestrations are accepted in BPEL, which is considered a standard for describing such service behaviour descriptions. Where other work has concentrated on specifying formal algebraic notations for service compositions, we provide a graphical interface assistance so that the user does not have to learn these sometimes complex notations. Verification properties are specified by input of deployment configurations diagrams in UML XMI format. The approach complements existing verification and validation properties through an integrated, simplified interface. Results (traces to violation) are displayed back to the user in an accessible form, in this case using MSC notation.

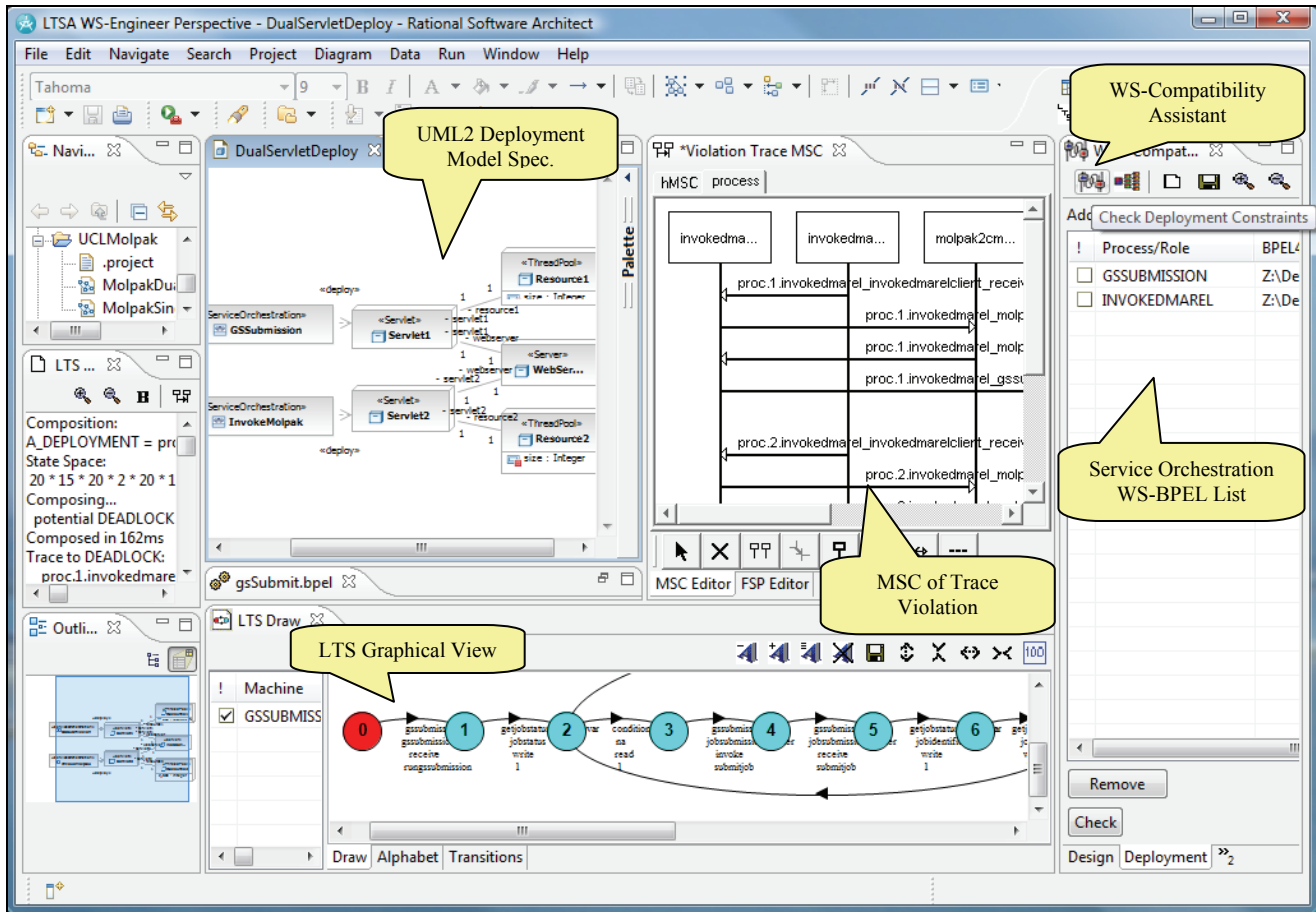


Figure 8 Rational Software Architect with WS-Engineer for Service Composition and Deployment Analysis

### 6.3. Early Payback

Early payback is a key objective of the approach. Through the motivation for this work, the aim is to support answering questions highlighted by the behaviour of web service compositions and on consideration of the orchestrations which these compositions may interact. Clearly, the benefits of using such an approach will require early feedback to the developers so that greater assurance can be given, in both design and implementation activities, as earlier as possible. This was achieved by separating the tool between design and implementation, and consolidating output to provide another view for analysis.

### 6.4. Orientation towards error detection

The essence of the approach is to highlight inconsistencies between interactions specified in composition implementations against that of those specified for resource activities in deployment configurations. The aim is not to clarify notational and specification semantic correctness, although that can be achieved to a degree by user validation through animation. The assumption is that correctness of implementations is a given attribute of the inputs submitted for observing errors

against design specification scenarios, and as such, this provides orientation of our approach towards error detection rather than correctness of these artifacts. We also do not believe in forcing a new methodology upon developers by way of the tool, but aid in various methodologies for the tasks that must be undertaken regardless of the actual steps of a methodology e.g. verification and validation can be undertaken in either analysis, design, implementation or maintenance stages of a development lifecycle.

### 6.5. Assumptions and Limitations

The approach is flexible to handle any configuration for deployment diagrams which can be exported as standard UML XMI. However, prior to undertaking the approach, the engineer must configure a resource map (discussed in section 4). To construct a resource map is a simple container with a list of activity types (invoke, receive, reply etc) and the related resource action (get or put). To acquire knowledge of this mapping is more complicated and relies on knowledge of the implementing service orchestration engine. For example, the case study used ActiveBPEL engine for runtime execution. We studied the

architecture and logs of the engine to determine this mapping. When this knowledge is stored however, we can assist the engineers by providing resource map lists to select from for different engine profiles. An additional constraint is posed in the model on specifying a resource pool size. In the case study we counted the activities in the orchestration to determine a suitable starting size. This, however, mirrors the practice of setting up the physical configuration of service hosts, but our design verification can be used to determine if such a setting would cause undesirable effects at runtime. Also, inherent in model checking is a scalability issue relating to the degree of abstraction required, the state space of the models generated and the type of analysis performed. The complete model discussed previously was based upon a number of abstractions for workflow and resource modelling. Firstly, in the orchestration mapping, the case study workflows used 200 parallel invocations of the same requirement and thus we were able to abstract to 2 client invocations (of a single invocation type) to check concurrency on a smaller scale. This requires further analysis of state space usage and the approach, to find limits. Also the example concentrated on using threads as an example resource constraint yet there are resources which affect behaviour and resource usage. This includes memory pools for example. We are also keen to explore other resource types and use with the same approach.

## 7. Conclusions

In this paper we described an implementation of an approach to assist service engineers in checking the behaviour of service compositions with deployment constraints by model-checking service orchestrations assigned to various elements of a deployment configuration. There are a number of limitations in the approach which can be made more efficient (for example specifying the behaviour activity and resource action mappings) and we are keen to improve the tool implemented to support this. The approach has been fully implemented in the WS-Engineer plug-in which is part of the LTSA Eclipse analysis suite. This work has been funded partly by the EU project IST-2005-16004 (SENSORIA). The tool suite is available for download at: <http://www.doc.ic.ac.uk/ltsa/eclipse>.

## 8. References

- [1] H. Foster, W. Emmerich, J. Kramer, J. Magee, D. Rosenblum, and S. Uchitel, "Model checking service compositions under resource constraints," in *Proceedings of the the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering (ESEC/FSE)* Dubrovnik, Croatia: ACM, 2007.
- [2] International-Business-Machines, "IBM Rational Software Architect (v7.0.0.4)". Available from: <http://www-306.ibm.com/software/awdtools/architect/swarchitect/>, 2007.
- [3] Object-Management-Group, "OMG The Unified Modelling Language (OMG UML) v2.1.2," November 2007.
- [4] A. Alves and et-al., "Web Services Business Process Execution Language (WS-BPEL)," Organization for the Advancement of Structured Information Standards (OASIS) 2007.
- [5] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture (WS-A) - W3C Working Group Note 11 February 2004." vol. 2004: W3C Web Services Architecture Group, 2004.
- [6] C. Ouyang, W. v. d. Aalst, S. Breutel, M. Dumas, A. t. Hofstede, and H. Verbeek, "Formal semantics and analysis of control flow in ws-bpel (revised version)," 2005.
- [7] R. Hamadi and B. Benatallah, "A petri net-based model for web services composition," in *3rd IEEE International Conference On Web Services (ICWS)* San Diego, CA, USA, 2004.
- [8] G. Salan, L. Bordeaux, and M. Schaerf, "Describing and Reasoning on Web Services Using Process Algebra," in *3rd IEEE International Conference On Web Services (ICWS)* San Diego, CA, USA, 2004.
- [9] X. Fu, T. Bultan, and J. Su, "WSAT: A tool for Formal Analysis of Web Services," in *16th International Conference on Computer Aided Verification (CAV)*, Boston, MA, 2004.
- [10] A. Chakrabarti, H. L.D., and M. Stoelinga, "Resource interfaces," in *3rd International Conference on Embedded Software (EMSOFT)*, 2003.
- [11] C. Tofts, "Efficiently Modelling Resource in a Process Algebra (hpl-2003-181)", Technical Report, Available at <http://www.hpl.hp.com/techreports/2003>, 2003.
- [12] J. Haymann, "The Application of a Resource Logic to the Non-Temporal Analysis of Processes Acting on Resources (hpl-2003-194)", Technical Report. Available at <http://www.hpl.hp.com/techreports/2003>, 2003.
- [13] W. S. Lee, A. S. McGough, S. Newhouse, and J. Darlington, "A Standard Based Approach to Job Submission through Web Services," in *Proc. of the UK e-Science All Hands Meeting*, Nottingham, UK, 2004, pp. 901-905.
- [14] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, "Grid Service Orchestration using the Business Process Execution Language (BPEL)," *Journal of Grid Computing*, vol. 3, pp. 283-304, 2005.
- [15] H.Foster, S.Uchitel, J.Magee, and J.Kramer, "WS-Engineer: A Tool for Model-Based Verification of Web Service Compositions and Choreography," in *IEEE International Conference on Software Engineering (ICSE)*, Shanghai, China, 2006.
- [16] J. Magee and J. Kramer, *Concurrency - State Models and Java Programs*: John Wiley, 1999.
- [17] The-Apache-Software-Foundation, "Apache-Tomcat 5.5". Available at: <http://tomcat.apache.org/>, 2005.
- [18] Active-Endpoints, "ActiveBPEL Engine". Available at: <http://www.activebpel.org>, 2005.