Leveraging Eclipse for Integrated Model-Based Engineering of Web Service Compositions

Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer
Imperial College London
180 Queen's Gate
London, SW7 2BZ, United Kingdom
+44 (0)20 7594 8298
{hf1,su2,jnm,jk}@doc.ic.ac.uk

ABSTRACT

In this paper we detail the design and implementation of an Eclipse plug-in for an integrated, model-based approach, to the engineering of web service compositions. The plug-in allows a designer to specify a service's obligations for coordinated web service compositions in the form of Message Sequence Charts (MSCs) and then generate policies in the form of WS-CDL and services in the form of BPEL4WS. The approach uses finite state machine representations of web service compositions and service choreography rules, and assigns semantics to the distributed process interactions. The move towards implementing web service choreography requires design time verification of these service interactions to ensure that service implementations fulfill requirements for multiple interested partners before such compositions and choreographies are deployed. The plug-in provides a tool for integrated specification, formal modeling, animation and providing verification results from choreographed web service interactions. The LTSA-Eclipse (for Web Services) plug-in is publicly available, along with other plug-ins, at: `http://www.doc.ic.ac.uk/ltsa`.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided Software Engineering (CASE).* D.2.4 [**Software Engineering**]: Software/Program Verification – *model checking, correctness proofs, Validation.*

General Terms

Service Design, Implementation, Standards, Verification, Validation.

Keywords

Eclipse plug-in, Web Service Composition and Orchestration, Web Service Choreography, Verification, Validation, Model Checking

Eclipse Technology eXchange (ETX) 2005 at Object-Oriented Programming, Systems, Languages and Applications 2005, October 16–17, 2005, San Diego, California, USA.

1. INTRODUCTION

The practice of engineering web services currently focuses on the technical implementation of service functionality in a number of development language environments (such as Java and .NET). There has been a growing presence of web service tools, aimed mostly at the developer, who can generate web service interfaces and deployment configurations for hosting services on these different environments, yet the service interactions are equally important in the consideration of how the clients (including other services) needs will be incorporated in to the design of service functionality. Our work to date [1-3] focuses on providing an approach and tool which facilitates the design of appropriate service interaction specifications, verifying implementations of these specifications and generating representations in the standards for web service orchestrations and choreography. We achieve this through the provision of editors and views for analyzing the scenarios in service interactions and by verifying properties of the models produced through interaction specifications. This paper is organized as follows. In section 2 we describe the design goals by considering the aspects of web service compositions in relation to the web service standards. We then describe in section 3, a path of engineering service compositions in the tool from design through to maintenance. Section 4 details the approach to plug-in architecture for Eclipse, with Section 5 providing a brief example. Section 6 is an evaluation of the plug-in and Section 7 concludes the paper with opportunities highlighted as part of this on-going work.

2. DESIGN GOALS

Web Service behaviour analysis consists of analysing two aspects of web service architecture style. The web service formally exhibits its identity and permissible interactions through definition in the Web Service Description Language (WSDL). Within the implementation for a web service however, the behaviour of its interactions is defined. The coordination of a service's behaviour is formed from the basic operations of requesting, receiving a new request, replying to a service or receiving the reply from a request and this forms the basis for service analysis for its interaction behaviour. Standards elaborate the specification of how, what and when these interactions can occur. The layers above the basic service are described through compositions, choreography, transactions and policies. The main goal of our tool is to provide an integrated environment which provides service clients, designers and engineers features to assist in the development and maintenance of web service compositions. The design of the tool builds editors and views for service interaction models, which can in turn be translated in to the Finite State Process (FSP) notation. Using the

Labelled Transition System Analyser (LTSA) [4] libraries, the FSP notation can be compiled into Labeled Transition Systems (LTSs) which in turn can be verified for correctness (using the formal software process techniques of liveness and reachability analysis). Additionally, the service interaction models can be translated to the web service standards for composition and choreography, namely the Business Process Execution Language (BPEL4WS) [5] and Web Service Choreography Language (WS-CDL) [6]. Together, these standards form the basis for describing the behavior of orchestrated web services and the interaction policy for multi-party process transactions.

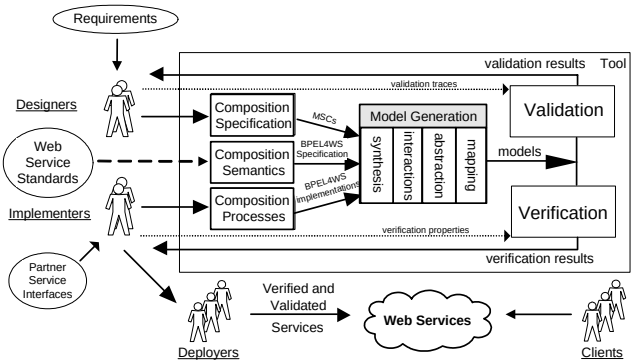


Figure 1 A Rigorous Approach to Engineering Web Service Compositions

We now describe how this approach is undertaken by the user, from design through build, and on to verifying the implementation against design specifications.

3. THE USER EXPERIENCE

3.1 Design Phase

We believe the design phase of engineering web service compositions consists of two aspects. Firstly, the required service interaction behavior is specified, highlighting where and in which order the interactions between two or more service partners must be sequenced. In our approach the designer specifies the partners and interactions to fulfill this composition by way of building Message Sequence Chart (MSC) scenarios for the different sequences of interaction that are possible. The plug-in includes an MSC editor, which provides basic and high level MSC editing capability. The second part of designing the service compositions is to model the scenarios and validate the sequences possible through an animation of the interactions that can occur. This facility is provided in the tool by an LTS Animator tool. Indeed, requirements engineers must not only elicit and document requirement scenarios, but also validate that these are indeed what stakeholders want [7]. The technique of simulation through animation is an effective validation technique, whereby in its simplest form, stakeholders can step through sequences of events dictated by a behaviour model [8].

3.2 Build Phase

The build phase provides the service engineer with tools to generate, enhance and analyse documents implementing the service process and the service policy. Our approach currently supports analyzing BPEL4WS and WS-CDL, yet as other standards emerge these can be incorporated in to the approach. Additionally we are working on generating templates of standards based implementations for both choreography and service implementation. The implementation of service compositions undertaking one or more roles within this service choreography is undertaken by a service engineer who defines the sequences of interactions within each service role as part of the choreography enactment. Interaction processes can be implemented in the emerging standard of the BPEL4WS which provides a workflow language schema to implement such a composition. In the approach these compositions are also translated to the FSP algebra and compiled in to LTS models. Through a process of abstraction and mapping, the composition interactions are combined to provide an architecture model of the cooperating services.

3.3 Verification and Validation

Given the outputs of the architecture models from the design and build phases, the users of the approach can perform verification against a series of specified properties. For example, the designer can verify that if there are any implied scenarios (scenarios which may be unexpected given the initial specified sequences) in the interaction specifications. From the perspective of the service engineer, they can also undertake analysis of whether there are any deadlocks present on a reach ability analysis of the implementation. Such situations could occur if conflicting dependencies between interactions have been configured by mistake or misaligned. Related to this is an analysis of interactions between compositions, which we term “compatibility” for the available sequences of interactions to be fulfilled. This analysis takes the web service communication model (a port connector for request, receive and reply) and checks that two or more compositions may fulfill each interaction cycle. The choreography model can then be compared with two or more compositions such that the interactions fulfill partner roles as defined in the choreography specification (described in the design phase). One clear use for this is that that of reusable components, as an initial requirements baseline is considered when the first deployment occurs. The expectations of these components will quickly be exhausted as new requirements and further functionality are required by additional partners in a composition [9, 10].

4. ECLIPSE PLUG-IN DESIGN

4.1 From Standalone to Plug-in

Using the Eclipse framework opens the potential to link the tool with a network of other Eclipse plug-in contributions and aims to simplify the number of different, bespoke tools used in software engineering as a whole. There were several reasons why we sought to leverage the Eclipse Integrated Development Environment (IDE) for our work and develop a IDE based tool rather than extending the previously standalone LTSA tool. Firstly, a growing number of editors have been released to support a number of different languages and specifications (for example, Java, C#, C++, BPEL4WS etc) irrespective of actual technology deployment environment. Our approach required an IDE which was flexible to multiple editors and views working closely together. Secondly, the notion of providing *extension points* promotes contributing your plug-in not only to increase the number of available plug-ins, but also work closely with other contributors to enhance the overall engineering experience by plug-ins working together. Indeed,

amongst these contributions are commercial BPEL4WS graphical editors (we currently only provide a basic XML editor), although the reader is invited to browse plug-in web sites as the list of contributors is continuously expanding. To migrate an original LTSA based plug-in to the Eclipse environment consisted on rebuilding the model, views and controller pattern using the Eclipse Plug-in development environment. There were a few issues that we attributed to challenges when moving from standalone to an extended plug-in version of the LTSA tool. This consisted of

- **Communication:** To provide a consistent and expandable mechanism to support cross editor and view updates. As changes occur to document, reflect this in any associated views. Additionally, support cross plug-in collaborative development.
- **Job Performance:** Enabling threaded jobs in translation, synthesis, compilation and process analysis. Long running jobs should not restrict other work being undertaken, and should provide continuous feedback to the user.
- **Graphics Conversion:** The original draw views were written in the AWT/SWING API. The effort required to migrate this to SWT support was unknown before work began, and before the AWT/SWING to SWT Bridge was included in general release (i.e. org.eclipse.swt.awt.SWT_AWT).
- **Perspectives:** Building appropriate views for the user, given designer, service validation (by client) or service engineer roles.
- **UI Actions:** Multiple or central locations to support common actions on documents or process parts. For example, compiling the FSP from single or multiple sources (BPEL4WS, WS-CDL).

4.2 Plug-In Architecture

As a first step, we designed an architecture supporting the editor, model and views to enable the layered engineering approach to be automated and to identify the areas where the issues above would be most apparent. The architecture of the tool consists of three layers, as illustrated in Figure 2.

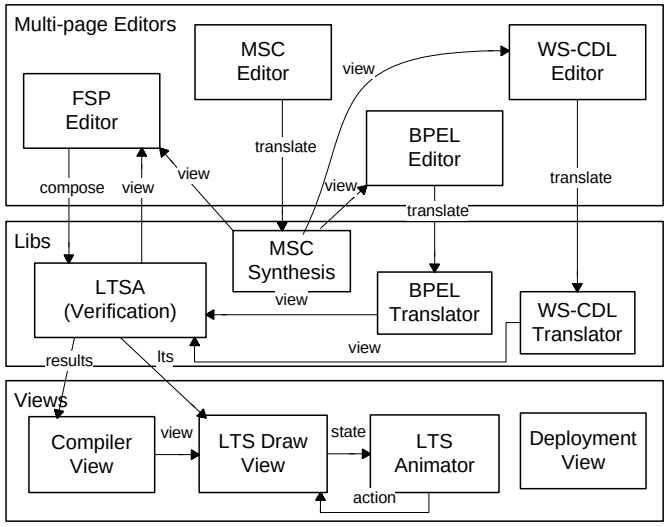


Figure 2 Plug-In Architecture of Editors, Models and Views

Firstly, a set of editors provides document management for MSCs, BPEL4WS and WS-CDL specifications. Each of these editors has a related model function library in the second layer of the architecture, to support synthesis or translation of the source document into an FSP process model or in the case of the MSC synthesis, additionally to generate templates for BPEL4WS or WS-CDL specifications. By way of the LTSA modules, the web service specifications can be represented back to the user in the form of the FSP editor, from which analysis can be performed. The third layer contributes additional views on the results of modeling and compiling the source documents or FSP respectively. For example, compilation and analysis of the FSP by way of the LTSA module is presented back to the user in the Compiler View (also known as Output). Secondly, the compiled model can be viewed as a graphical LTS in the Draw View and animated through trace runs for validation purposes. We also envisaged providing a deployment view once verification and validation has been satisfied to further facilitate the service engineering cycle.

4.3 Model and Editor Views

The FSP, MSC, BPEL4WS and WS-CDL Editors all extend the *MultiPageEditorPart* class as part of the included Eclipse plug-in development environment. The documents behind these editors are XML based, except for the FSP notation which is textual yet still easily machine readable. Extensions to support calling the appropriate editor are easily configurable in the *plugin.xml* deployment config file. The editor content is scanned on an "input rest" (i.e. after there is a delay in user editing interactions) and upon document restore or save actions to provide useful editor functions, such as syntax highlighting. A full parsing of source is however, performed on compilation of the FSP source, whereby an *outline* view content is updated with a breakdown of an FSP document. This includes a list of compositions (such as specified parallel processes), a list of basic processes (a process or a sequence of processes). The engineer is able to build one or many web service compositions which aids in integrated enterprise service decomposition. For each composition selected, the engineer can either translate a single composition (by way of a mechanical implementation of translation rules described in our earlier work) or compose multiple compositions for choreography and translate them in to FSP. The translation module is written as an independent module (itself potentially a web service), which takes as input one or more BPEL4WS or WS-CDL implementations and in turn, traverses the source building a representation model in FSP. Problems in translation or with parsing of documents are listed in the output view, as well as specific syntax problems added to the core Eclipse *Problems* view. The other useful feature was to have views with multiple tabs, using the core *Page* sub-part of a MultiPageEditor. As translations occurred, pages can be dynamically created or removed.

Each of the views in the plug-in extend the *ViewPart* class included in the eclipse core libraries. The compiler output (used in compilation of the FSP and in translation of BPEL4WS and WS-CDL to FSP) is currently based upon a simple extension of the eclipse *TextEditor* class. Compilation runs a threaded task. We utilised the recently included *org.eclipse.core.runtime.jobs* to wrap existing single threaded compilation and analysis with threaded tasks and included the progress monitor to support feedback to the user of job progress. We were limited however, in the ability to ascertain at what point the compilation had reached (the number of process model states is unknown at the time the job begins). Results of checks provide implementers and designers with useful

details such as missing interaction cycles (e.g. a missing receive or reply action). An output view summaries actions undertaken by the LTS compiler, and reports on property violations, such as deadlock, liveness or other safety properties.

The relationship that editors and views have is built around the core listeners added to each. This is by no means a single way relationship. For example, in the *Outline* view, the contents of the outline are updated whenever a parse occurs against the FSP source code. However, when a user clicks a composition or process selected in the outline, the editor moves the text cursor to the line location of the beginning of the source for the selection made. Leveraging the flexibility of event handling in Eclipse requires the developer to think about possible clashes in events, and building safe guards around these to provide a consistent view to the user. The main LTSA tool also supports trace Animation, process Alphabets and process Transitions in the LTS Draw view. We are continuing to migrate the full functionality of the LTSA tool across to the Eclipse environment, yet this subset has already supplied those necessary for significant web service composition analysis.

4.4 Challenges and Issues

Relating back to our initial thoughts on challenges and issues of the plug-in, our experience has shown that the core LTSA Java modules could be successfully imported into the Eclipse plug-in development environment however; rebuilding the graphical view modules has required some changes, particularly when moving the LTS Draw view to the SWT API as discussed previously. Aside from these differences however, current application and view migration has mapped conveniently onto the standard views provided by the Eclipse framework (Editor, View, Outline, Console etc). We found that the difficulty in extending plug-ins to communicate, without prior knowledge of the internal class structure of a deployed plug-in, limited our scope to leverage the work of others. For example, we would like to be able to communicate with more enhanced graphical BPEL4WS editors, yet there is no simple way to achieve this presently. Additionally we found that wrapping tasks with the core eclipse job classes was simple in the activity, yet the most appropriate way to use this would involve rewriting the task itself to exhibit useful progress information.

5. AN EXAMPLE

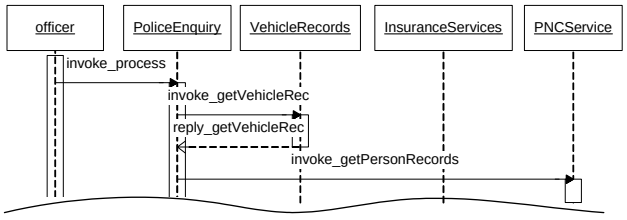
As a brief example, we have recently worked closely with the UK Police Information Technology Organisation (PITO) on scenarios for police officer enquiry services. The scenarios consist of a series of enquires (such as vehicle insurance, vehicle registration, person, weapons registry etc) and the compositions of these enquires. Figure 3 includes an example process for one such scenario for these enquiries. With a sequence of interactions specified in an MSC, the translators can be used to generate FSP and BPEL4WS process statements and WS-CDL choreography rules by way of synthesizing the MSC specification, its components and the interactions (as illustrated in Figure 4). In BPEL4WS, a process is generated by considering the possible sequence and concurrency of interactions from the initial enquiry (the main process) and between other services. Each component may be considered a separate BPEL4WS process. The choreography is a series of interactions between two or more components in the MSC. Currently, the tool provides only limited generation of BPEL4WS and WS-CDL specifications, whereby single (basic) MSC scenarios are translated to a process. Complete MSC and BPEL4WS processes (directly built by an engineer) are however, translated fully to equivalent interactions models in FSP (by applying the semantics of FSP to each of the BPEL4WS and WS-CDL constructs). We are continuing to develop the synthesis of MSCs such that a series of scenarios is represented fully in the processes and choreographs generated. The processes can then be analyzed for correctness using model-checking techniques (such as against dead-lock and liveness properties). A full list of translation semantics from BPEL4WS to FSP are presented in [2].

6. RELATED WORK

There are a number of initiatives to utilise Eclipse for modeling and then generating web services compositions including [11] and [12]. We focus on modeling web services as reusable components from the perspective of assessing multiple client usage scenarios. We achieve this by using formal software process verification techniques and providing the user with validation analysis to ascertain if requirements have been implemented appropriately. The related work can be seen as opportunities to contribute our plug-in such that their work could potentially benefit from this analysis.

7. CONCLUSIONS AND FUTURE WORK

Firstly, we wish to continue describing compositional behaviour by elaborating on the wider choreography aspects of partnered services, and are also seeking to model the requirements of distributed resources for service compositions (such as impact of number of instances and type of requests between compositions). This also includes considering fault, compensation and transactional integrity within and between distributed processes. As part of this work we are aligned closely with consortiums, such as the W3C, on their work with choreography architectures and specifications. It is anticipated that the result of their work could be incorporated into our approach and the plug-in to provide an extension to the choreography elements we have considered thus far.




```
// build interaction sequences for PoliceEnquiry Process
// Scenario 1.
P1 = (receive_officer_pito1_process -> END).
P2 = (invoke pito1 vehiclerecords getvehiclerec -> END).
P3 = (reply_vehiclerecords_pito1_getvehiclerec ->END).
SEQ1 = P2; P3; END.
P4 = (invoke pito1 pncservice getpersonrecords -> END).
P5 = (reply pncservice_pito1_getpersonrecords ->END).
SEQ2 = P4; P5; END.
.....
MAINSEQ1 = SEQ1 ; SEQ2; .....
PITO1 VEHICLE RECORDS SEQ ; PITO1 NOMINAL ENQUIRY SEQ ;
PITO1 VEHICLE INSURANCE SEQ ; PITO1 ANPR ENQUIRY SEQ ;
PITO1_FINGERPRINT_ENQUIRY_SEQ ; PITO1_REPLYOUTPUT; END.
// Scenario 2.
.....
// compose sequences into parallel composition
||PITO1 Instance = (MAINSEQ1 || MAINSEQ2 || .....).
||PITO1_BPELModel = (MAINSEQ).
```

Figure 3 Example synthesis of MSC (top) to FSP (below)

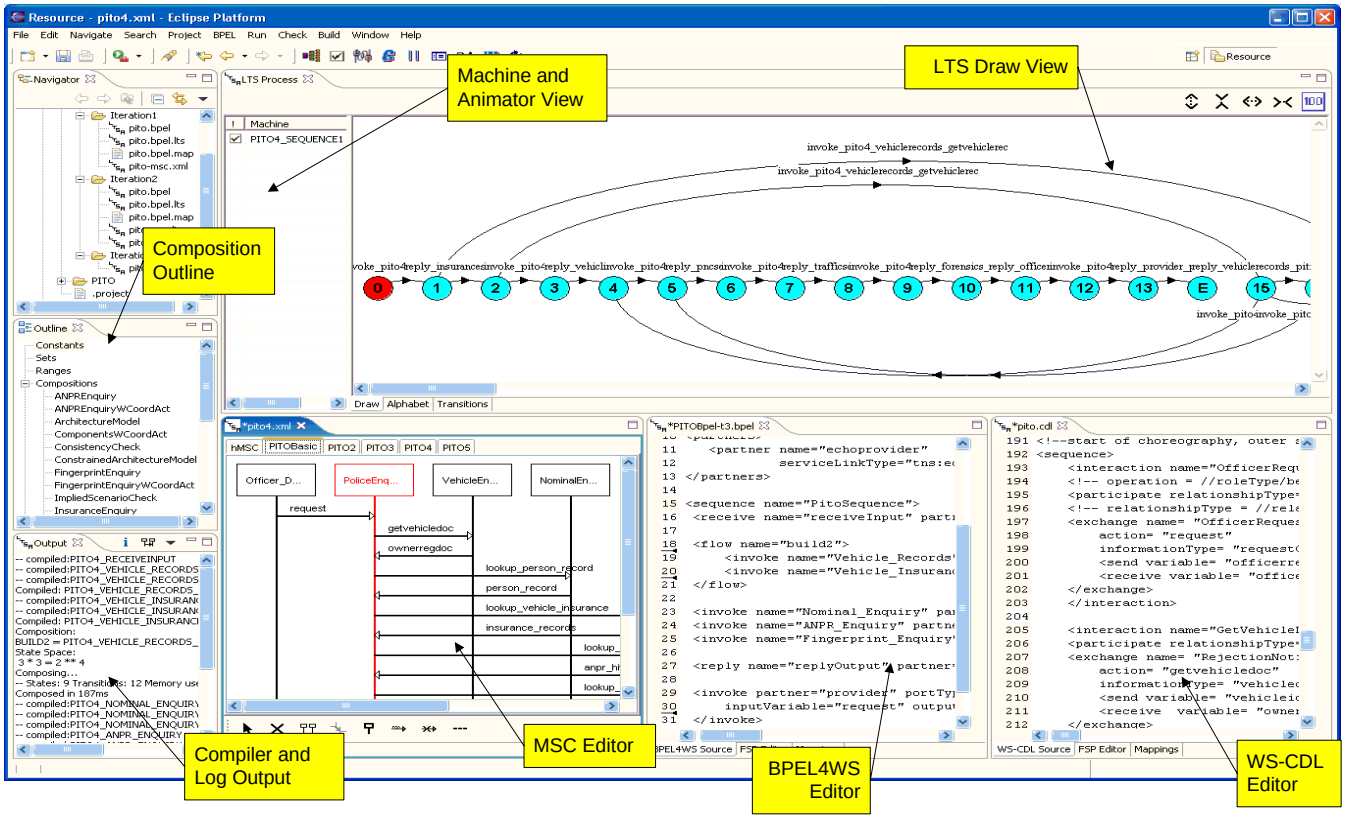


Figure 4 LTSA-Eclipse: Eclipse IDE Perspective of Editors and Views

We are also evaluating the use of core Eclipse graphical modeling plug-ins, such as the Eclipse Modeling Framework (EMF) [13] that may replace the custom MSC classes we have developed and for translation between graphical and textual notations the Meta Object Facility (MOF) [14]. The authors would like to acknowledge that this research was supported, in part, by the STATUS ESPIRIT project (IST-2001-32298) and by an IBM Eclipse Innovation Grant (EIG 2005).

8. REFERENCES

[1] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility for Web Service Choreography," presented at 2nd IEEE International Conference on Web Services (ICWS), San Diego, CA, 2004a.

[2] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions," presented at Eighteenth IEEE International Conference on Automated Software Engineering (ASE), Montreal, Canada, 2003a.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Tool Support for Model-Based Engineering of Web Service Compositions," presented at 3rd IEEE International Conference on Web Services (ICWS2005), Orlando, FL, 2005.

[4] J. Magee and J. Kramer, *Concurrency - State Models and Java Programs*: John Wiley, 1999.

[5] 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," 2004.

[6] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, "Web Services Choreography Description Language Version 1.0 - W3C Working Draft 17 December 2004," 2004.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," presented at International Conference on Software Engineering (ICSE'00), Limerick, 2000.

[8] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "Fluent-Based Animation: Exploiting the Relation between Goals and Scenarios for Requirements Validation," presented at Requirements Engineering (RE'04), 2004.

[9] J. Yang and M. P. Papazoglou, "Service Components for Managing the Life-Cycle of Service Compositions," *Information Systems*, 2003.

[10] M. Larrson and I. Crnkovic, "New Challenges for Configuration Management," presented at 9th Software Configuration Management Workshop, Toulouse, France, 1999.

[11] S. Iyengar, "Business Process Integration Using UML and BPEL4WS," presented at XML Conference and Exposition 2003, Philadelphia, PA, 2003.

[12] F. Curbera, M. J. Duftler, R. Khalaf, N. Mukhi, W. A. Nagy, and S. Weerawarana, "BPWS4J: A platform for creating and executing BPEL4WS processes," Component Systems group, International Business Machines (IBM), 2002.

[13] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose, *Eclipse Modeling Framework*: Addison Wesley Professional, 2003.

[14] OMG, "Meta-Object Facility (MOF), Version 1.4," *Object Management Group*, 2002.