In Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering, Long Beach, CA, Nov 2005.

A Visual Language and Environment for Composing Web Services

Na Liu
Dept of Computer Science
University of Auckland, Private Bag 92019, Auckland, New Zealand
+64 9 3737599
karen@cs.auckland.ac.nz

John Grundy
Dept of Electrical & Computer Eng
University of Auckland, Private Bag 92019, Auckland, New Zealand
+64 9 3737599
john-g@cs.auckland.ac.nz

John Hosking
Dept of Computer Science
University of Auckland, Private Bag 92019, Auckland, New Zealand
+64 9 3737599
john@cs.auckland.ac.nz

ABSTRACT
Implementing complex web service-based systems requires tools to effectively describe and co-ordinate the composition of web service components. We have developed a new domain-specific visual language called ViTABaL-WS and built a prototype design tool to support modelling complex interactions between web service components. ViTABaL-WS uses a "Tool Abstraction" metaphor for describing relationships between service definitions, and multiple-views of data-flow, control-flow and event propagation in a modelled process. The tool supports the generation of Business Process Execution Language (BPEL) definitions from a model, directly deploys a generated model to a workflow engine, and supports dynamic visualisation of a running BPEL process.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and techniques – *CASE, modules and interfaces*
D.3.2 [**Programming Languages**]: Language Classifications – *dataflow languages, design languages, high level languages.*

General Terms
Design, Languages

Keywords
Web services, tool based abstraction, visual languages.

1. INTRODUCTION
Web services are reusable, extensible, platform- and language-independent components that are used over web protocols. An abstract definition of a web service contains two parts: messages and operations [12], each service described using the Web Services Description Language (WSDL). Running web service operations are bound to ports and run on a host. Web services composition is an approach that integrates individual services to make up a web service-based distributed system. A web service composition language (either textual or visual) is needed to specify a composite web service, using existing service components defined or looked up from a services registry. The composed web service can then be described using WSDL, registered and invoked, and thus added to the network as a new web service component.

One common web service composition language is the Business Process Execution Language for Web Services (BPEL4WS [8]), an XML-based service composition language. It describes web services compositions, or orchestration, by defining a set of service partnerships and structured invocation schemes. It also supports specifying concurrency and transaction failure recovery schemes for composed web service components.

Various visual modelling notations have been developed to support web service composition. UML state-charts can be used to specify implementation aspects of a service composition [1]. These incorporate event handling schemes where states represent services, transitions are constrained by Event-Condition-Action rules, and an occurrence of an event fires a transition to execute a target action. UML-WSC [11] uses class diagrams with stereotypes to model static structure and activity diagrams to model dynamic aspects of web service compositions. Service states call operations from components and transform states perform structural transformation on messages. Message Sequence Charts (MSCs) are compiled into a Finite State Process notation (FSP) to concisely describe and reason about concurrent programs [3]. Petri-Nets have been used to model both offline analysis tasks, such as web service composition, and online execution tasks, such as deadlock determination [6], [9]. These approaches describe the capabilities of web services in terms of a first-order logic language. Biopera Flow Language [10] is a generic visual flow language for coordinating software components, with a development tool tailored for web service composition. This focuses on data flow, execution sequence and fault handling and all can be specified with a simple visual syntax. The Web Service Modelling Framework [2] is a methodology for describing and developing web services and their compositions. The integration framework defines a conceptual model for the web services integration (complex web services) and provides services for mediating differences in data structures and message exchange patterns among services.

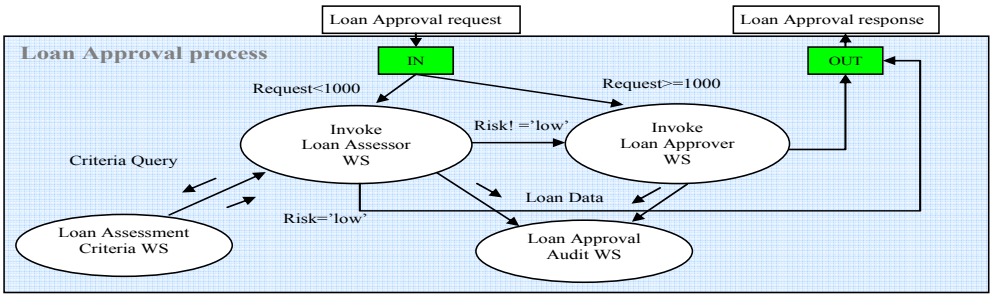

Figure 1. Conceptual model of the loan approval process

Most of these current approaches to modelling web service compositions lack full modelling capability: i.e. are not able to model all types of operations (one-way, request-response, solicit-response, notification). A common drawback is that a web service interface can not be fully expressed; some model web services operations only; and some can not model invocation constraints in control flow. Most use static binding rather than event-based mechanisms to integrate services. Many cannot separate or combine control-flow and data-flow for modelling.

2. ViTABaL-WS

Consider a simple loan approval process, as used in the description of IBM's Business Process Web Service for Java (BPWS4J) process execution tool [7]. This loan approval process is composed of two main web services: a Loan Assessor web service and a Loan Approval web service. As illustrated in Figure 1 when a loan request is received, the new Loan Approval process firstly needs to determine whether the requested amount of the loan is under one thousand dollars or not. If the amount is under one thousand dollars, the Loan Assessor web service is invoked; otherwise the Loan Approver web service is invoked. After the Loan Assessor web service is invoked, the process continues by determining whether the risk for the request is low or high: if the risk is high, the control flows to the Loan Approver web service; otherwise an approval message is generated as the response to the user's loan request. Additional web services might also be used e.g. to provide Loan Assessment Criteria (from a persistent storage mechanism), and to record a Loan Approval Audit trail (storing the loan and approval information in a persistent form for later reporting). Relationships between services in such business process models can become very complex: some send messages and wait for replies; some send messages and continue execution; some provide data while others consume it; synchronisation between concurrently executing services may be needed; service failure may occur and needs to be handled appropriately; and transactional behaviour may be required over services.

We aimed to specify a language and tool that meet the following key requirements:

- Uses a visual metaphor for composing web services that fits the users' mental models of service interaction;
- The visual language for composition must be able to specify: web service interfaces, i.e. abstract message types and operations; variables; and different types of connections (i.e. data flow, control flow, and event flow) between web services in a process;
- The support tool should permit modelling of specifications using the metaphor/visual language; generation of WSDL and BPEL4WS (or other executable business process modelling languages), and easy deployment of generated process models to 3rd party process engines, e.g. BPWS4J;
- The support tool should permit visualization of running systems by annotating high-level visual specification views from events generated by the process engine, for debugging of compositions and understanding of others' specifications.

We chose to use the "Tool Abstraction" (TA) paradigm [4], [5] as our metaphor for web service compositions and to support reasoning about different relationships between compositional primitives. The TA paradigm is a message propagation-centric approach describing interconnections between "toolies" (the encapsulation of functions) and "abstract data structures" (ADSs: the encapsulation of data) which are instances of "abstract data type" (ADTs: typed operations/messages/events). Connection of toolies to other toolies and ADSs is via typed ports. The TA paradigm supports modelling data flow, control flow and event flow relationships. Reusability, extensibility and expressiveness are key advantages possessed by TA [4]. We have found that the TA paradigm is well suited for web services composition domain by specifying an abstract model involving a series of co-ordinated invocations to web services operations. We adapted our earlier work ViTABaL [5] to develop a new visual language and environment, ViTABaL-Web Services (ViTABaL-WS), specializing the ViTABaL visual composition language to the domain of web services composition. It supports modelling of both event-dependency and dataflow in designing complex web service compositions using a visual notation.

Figure 2 shows various ViTABaL-WS diagrams illustrating examples of compositional primitives in the Tool Abstraction paradigm. Toolies (web services - shaded, green ovals) encapsulate data processing and interact with each other through both direct and indirect operational invocations using shared data structures (message ADT instances: rectangular, shaded icons); and event-driven dependencies indicating state changes to a Data Store ADS (data storage service). A system of typed input and output ports on toolie and ADS services provide message sources and sinks. Services are wired together using these ports with ports supporting only certain kinds of connection and message ADTs. Messages generated by a service output port are distributed to connected web service input ports. Many interconnection schemes are supported including one-way flow, request-response, asynchronous flow, and subscribe-notify. Additional controls support conditional flow, dynamic type checking, synchronisation, iteration etc.

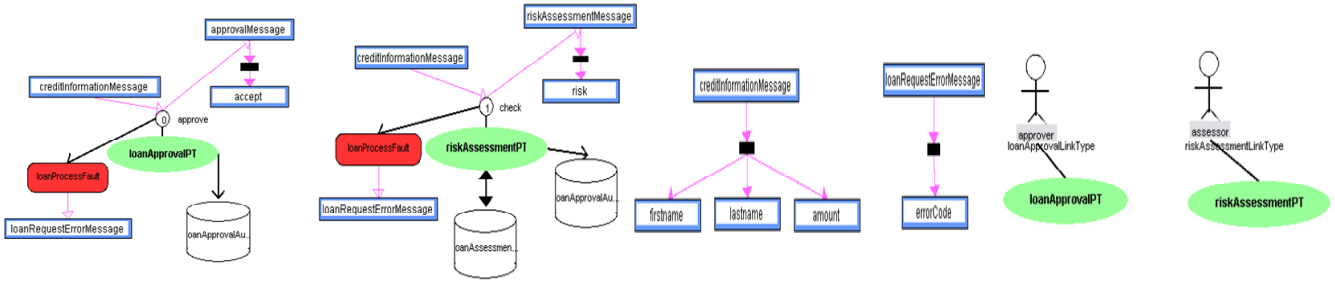

Figure 2. Examples of various webv service specifications from ViTABaL-WS.

ViTABaL-WS permits multiple views for complex processes and sub-processes, allowing a service in one process to invoke via ADS messages and ports another service or a sub-process. Different views allow both static specification of web service interfaces and dynamic specification of messages between processes in different views, with consistent references managed by the specification environment. Orthogonal views allow different kinds of interaction e.g. event-driven and data-flow, to be modelled separately if desired. The specified web services are linked together by composition rules enforced in the ViTABaL-WS tool.

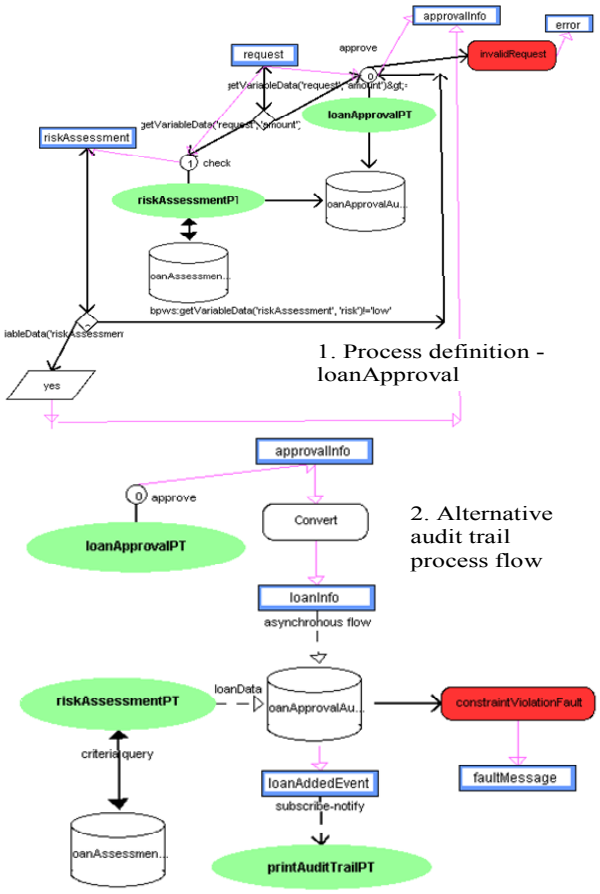

Figure 3.Process definitions in ViTABaL-WS.

Our exemplar process comprises two main information processing toolies ("PT" suffix): loanApprovalPT and riskAssessmentPT. The composite process defines roles performed by all participating services, i.e. "loan approver" service fulfils an "approver" role and "loan assessor" service fulfils an "assessor" role. Figure 2 (1) and (2) show the interfaces for the loanApprovalPT and riskAssessmentPT processing toolies. An abstract web service interface is visually represented using input/output dataflow links, parameter decomposition links, and transition links to support association of a toolie's web service port types and message ADSs. We attach operations to a port type to represent the port bindings of a web service. For example, in the "loan approver" web service definition in . Figure 2 (1) the "loanApprovalPT" toolie has one port providing an "approve" operation with "creditInformationMessage" as input message type (indicated by a data flow link with the arrow pointing to the operation) and "approvalMessage" as output message type (indicated by a data flow link with the arrow pointing out of the operation). The approvalMessage contains one message part, "accept" (shown by the parameter decomposition link). In the case of an error occurring when the toolie is invoked, the operation "approve" transits to the "loanProcessFault" fault handler (via a one-way operation link) which generates a fault message of type "loanRequestErrorMessage". The "loanApprovalPT" toolie may also invoke via a one-way operation a "loanApproval Audit" ADS to record an audit trail of approvals. Toolies may provide multiple ports for other toolies to bind too. Bindings may be data flow in/out, subscribe/notify event-based interaction, one-way async invocation, bi-directional synchronous invocation etc. Toolies may also have more than one fault handler for operations.

A business process model is built up by composing web service toolies using appropriate link types. Figure 3 shows the basic loan approval process. Note other overlapping views can be defined to add extra information about a process model e.g. extra: toolie links driven by event notification of asynchronous message flow; fault handling; message data storage/retrieval etc. The "loan approver" process defined in Figure 3 (1) expresses the semantics: the "loan approver" service receives a loan request. The process' control flows to a decision point, which retrieves the amount of the loan requested. The conditional is specified by labelling the outgoing links with an XPath expression specifying the comparison. If the requested amount is less than $1,000 the process control invokes the "risk assessment" service, else it flows back to invoke the "approve" operation of the "loan

approval" service. The "risk assessment" service takes the loan request as input and decides if the loan is a low risk. It retrieves loan criteria information from the "loanAssessmentCriteria ADS" tor use in the assessment task. If risk is low the loan is approved, otherwise the process model invokes the "approve" method in the "loan approval" service to do a more thorough check. Both toolies invoke data storage activity on the "loanApprovalAuditADS" to record an audit trail of approvals. Once a loan is either approved or rejected, an approvalInfo message is returned to the invoking client. Figure 3 (2) shows a different approach to generate an audit trail, with asynchronous flow from the generated "approval Info" message via an adapter converting its format to the "loan ApprovalAudit" service and generation of a "loan AddedEvent" notification subscribed to by a "print audit trail" service.

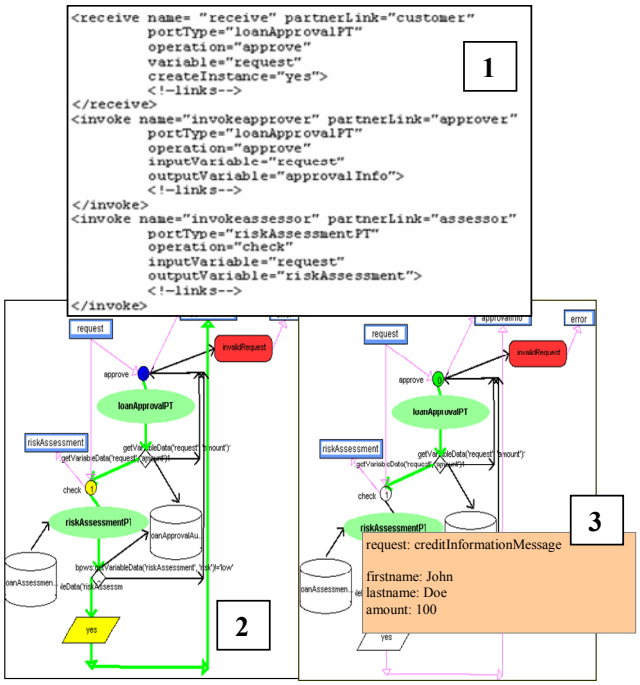

Figure 4. Generated BPEL and visualisations.

In order to execute our web service process model we translate our model into BPEL4WS. A BPEL4WS composition specification contains XML records specifying web services receiving messages, the service being invoked and reply message being generated (i.e. constructs <receive>, <reply>, <invoke>, <assign> etc). The ViTABaL-WS model contains TA-based modelling constructs that can be mapped onto BPEL4WS constructs. Processing and data storage/retrieval toolies map onto web services, with ADTs in ViTABaL-WS mapping onto BPEL4WS messages. Toolie ports map onto BPEL4WS ports with typing from ADT messages. Fault toolies and links to ports map onto BPEL4WS fault handlers. Synchronisation control, asynchronous message flow and subscribe/notify relationships in ViTABaL-WS map onto BPEL4WS process model script code to implement these behaviours. Concurrent operations in ViTABaL-WS map onto concurrently run BPEL4WS service invocations. Type checking toolies, conditional execution and iteration map onto BPEL4WS script to carry out these operations. Figure 4 (1) shows an example of some generated BPEL4WS code that is then run by the BPWS4J workflow engine.

Our dynamic visualization tool includes service invocation (by flashing the service representation node); invocation path into the service (by highlighting the path). Examples are shown in Figure 4(2) and (3). The user can double-click on a link or message and see its contents as XML. The traditional "debug and step into" metaphor is used to support step-by-step visualization. During each step of service execution, the states of all variables (messages) in the process are displayed in a debugging panel. Sub-processes invoked in the process are visualized similarly.

3. SUMMARY

We have developed several process models with our ViTABaL-WS tool and run these using the BPWS4J engine. We have carried out two evaluations of the visual language and support tool, one using Cognitive Dimensions and the other a usability user survey. These have demonstrated both the feasibility and suitability of our tool for developing web service composition models.

4. REFERENCES

[1] Benatallah, B., Dumas, M., Fauvet, M.C. and Rabhi, F. Towards Patterns of Web Services Composition, In Patterns and skeletons for parallel and distributed computing, Springer, 2003.

[2] Fensel, D. and Bussler, C. The web service modeling framework WSMF, Electronic Commerce Research and Applications, vol. 1, no. 2, pp. 113--137, 2002.

[3] Foster, H., Uchitel, S., Magee, J. and Kramer, J. Model-based verification of web service compositions. In Proc. 18th IEEE ASE, 2003, Montreal, Canada.

[4] Garlan, D, Kaiser, GE, and Notkin, D, Using tool abstraction to compose systems, Computer, 25(6) 30-8, 1992.

[5] Grundy, J.C. and Hosking, J.G. ViTABaL: A Visual Language Supporting Design by Tool Abstraction, In Proc IEEE VL'95, Germany, IEEE CS Press, pp. 53-60.

[6] Hamadi, R., Benatallah, B. A petri-net based model for web service composition, Proc 14th Australasian Database Conference, Adelaide, Australia, Jan 2003, CRPIT Press.

[7] IBM, Business Processes Web Services for Java, http://www.alphaworks.ibm.com/tech/bpws4j

[8] IBM, Specification: Business Process Execution Language for Web Services Version 1.1, http://www.ibm.com/developerworks/library/ws-bpel/

[9] Narayanan, S. and Mcllraith, S.A. Simulation, verification and automated composition of web services. In Proceedings of the 11th World Wide Web Conference, 2002.

[10] Pautasso, C. and Alonso, G. Visual Composition of Web Services, Proc IEEE HCC'03, Auckland, 2003, pp. 92-99.

[11] Thone, S., Depke, R. and Engels, G. Process-oriented, flexible composition of web services with UML, Proc ER-Wkshp on Conceptual Modeling Approaches for e-Business, Tampere, Finland, LNCS, 2002.

[12] W3C. Web Services Description Language (WSDL) 1.1, 2001, http://www.w3.org/tr/wsdl