Fall December 12, 2003

Facilitating Conversations among Web Services as Speech-act based Discourses

Karthikeyan Umapathy, *Pennsylvania State University*
Sandeep Purao, *Pennsylvania State University*
Vijayan Sugumaran, *Oakland University*

Facilitating Conversations among Web Services as Speech-act based Discourses

Karthikeyan Umapathy, Sandeep Purao
School of Information Sciences and Technology
Penn State University
kxu110@psu.edu, spurao@ist.psu.edu

Vijayan Sugumaran
School of Business Administration
Oakland University
sugumara@oakland.edu

Abstract

Web services composition is an emerging paradigm for enabling application deployment and integration within and across organizational boundaries. A landscape of languages and techniques for web services composition has emerged and is continuously being enriched. The ability to support conversations taking place between Web Services is an important area that is not yet fully addressed by the emerging standards. We suggest an approach to derive speech-act based message structures for conversation support that can complement existing standards.

1. Introduction

The web services technology is defined by loosely coupled, dynamically located software components deployed on the Web. Web services allow different applications, possibly written in different software languages and/or located on different parts of the Internet, to effectively find and communicate with each other. They can reside on different systems and can be implemented using vastly different technologies, but must be packaged and transported using standard Web protocols, such as XML and SOAP [Seth 2003]. The standardized, public invocation syntax provided by WSDL, UDDI, and SOAP along with ontology based DAML-S help service providers to describe their services and clients to search and invoke the required service.

A language such as BPEL4WS [BPEL4WS 2003] can be used to describe a process that may be realized with multiple services. For example BPEL4WS may describe a series of services to be invoked and the order in which they must be processed. However, these languages do not specify how peer-to-peer interaction between Web Services should take place. The ability to support a conversational mechanism for interaction between web services is an important area that is not yet addressed by emerging web service standards. A conversation mechanism for web services provides a more loosely coupled, peer-to-peer interaction model. These conversations involve multiple steps between parties, often involving negotiation between them. A peer-to-peer conversation model takes more of a third-person perspective, quite different from the extant standards of web service composition [Peltz 2003]. There are two proposals that attempt to support such a collaborative model - Web Service Choreography Interface (WSCI) and IBM's Conversational Support for Web Services (CS-WS). Neither, however, provides a mechanism for message generation to support peer-to-peer interaction that is grounded in existing theories governing interactions.

Our objective is to develop a methodology to generate messages that can facilitate peer-to-peer interactions in CS-WS. A speech-act theoretic approach underlies our work. In this paper, we provide a background of the problem in section 2, describe the methodology and formal specification for the messages in section 3, and conclude with a discussion of conclusions and future work in section 4.

2. Background

2.1. Web Services Technology Stack

Sleeper [2001] describes a web services technology stack with two layers labeled 'core' and 'emerging.' The first consists of protocols such as XML and SOAP. The second consists of WSDL, UDDI, WSFL and other business rules. Clearly, membership in these layers can vary with time. Burg [2001], on the other hand, makes a distinction between layers with the labels: syntactic and semantic. The

first consists of transport, network protocol and syntax. The second consists of ontology, content, communication and conversation. Neither view offers a complete picture of the different standards, which have evolved considerably. A more complete picture must take into account several layers including concerns such as security and conversations. Figure 1 shows our conceptualization of the Web Service technology stack. This view is essential to understand the level at which conversation support between web services may reside, and the assumptions that may underlie such an effort. For example, conversation support presupposes a process description level such as BPEL4WS or WSFL, and focuses on exchange of messages based on the order of sequence of interaction described in the process description level.

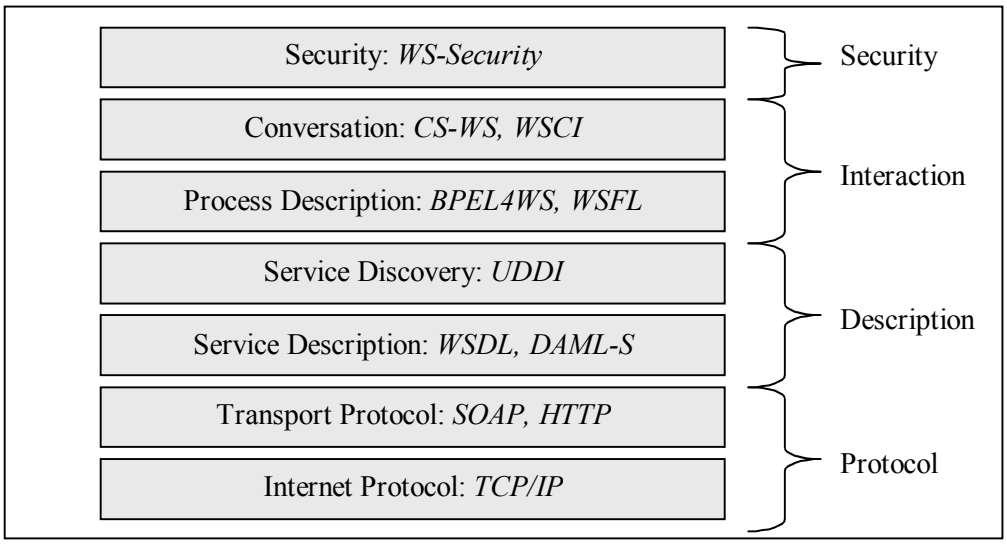

Figure 1: Web Services Technology Stack

2.2. Conversations and Policies: Vending Machines vs. Telephone Calls

Models for conversation support for web services imitate those for component interactions, in which components (applications, e-business, agents) are treated as autonomous, loosely coupled entities which interact by exchanging messages in a conversational context [Hanson 2002a]. In the conversation model, the interoperability technology consists of two distinct parts: (1) messaging and (2) conversation model. The conversation model governs formatting of messages sent, the parsing of messages received, and the sequencing constraints on exchanges of multiple, correlated messages. Typically, a separate subsystem mediates between the messaging system and the business processes [Hanson 2002b] to facilitate the conversation model. Figure 2 shows the conversation model for B2B message interactions.

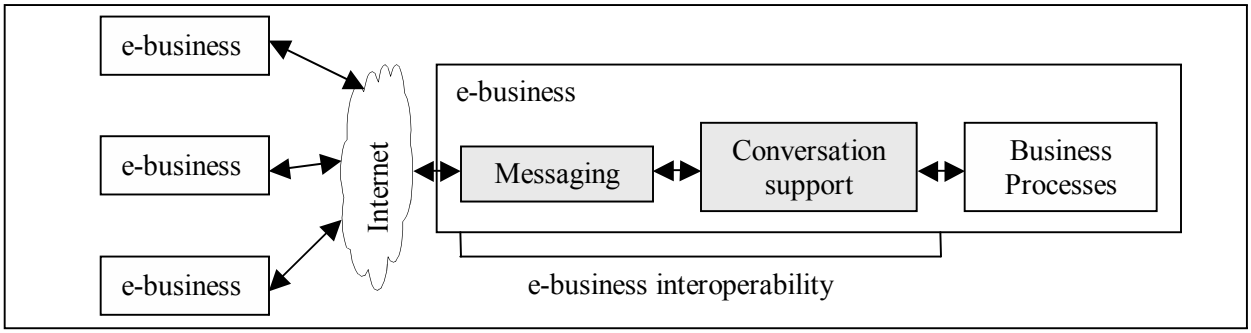

Figure 2: Conversation model for B2B message interactions [Hanson 2002b]

Current models for web service interactions are analogous to a vending machine, in which its WSDL port types and operations define the various buttons, levers, slots, and so forth, on the front of the machine. An inspection of the WSDL does not reveal that one must deposit money in the slot before

pushing any of the buttons. Web Service choreography languages such as WSCI [WSCI 2002] attempt to solve this problem in the same way: in effect, by giving instructions on which buttons to push in what order. CS-WS, on the other hand, replaces the "vending machine" model with a "telephone call" model. Two (or more) parties set up a conversational session, exchange various messages in the context of that conversation, and then, finally, close it down [CS-WS 2003]. Conversation models, thus, adopt a conversation-centric interaction. This means that messages are sent within an explicit conversational context that is set up on first contact, maintained for the duration of the conversation, and torn down at the end. Each new message in a conversation is interpreted in relation to the messages previously exchanged in that conversation. By adopting conversation-centric interaction, the business process interactions are represented as multi-step exchanges of correlated messages. A conversation model, thus, supports nesting and composition of conversational policies to provide a dynamic, adaptable, incremental, open-ended and extensible mechanism. The crux of the conversation model, however, remains the conversation policy (CP), which is a machine-readable specification of pattern of message exchanges in a conversation.

A conversation policy (CP) captures pre-programmed interaction patterns. A CP consists of a message schema, sequencing and timing information, which are best described by a state machine, in which the sending message (in either direction) is a transition from one conversational state to another. Conversation Policy XML (cpXML) is an XML dialect proposed by Hanson [Hanson 2002a] to represent such policies. cpXML is narrowly scoped. It restricts itself to describing message interchanges. It does not cover the way in which a conversation policy may be bound to the business logic, or the means by which it may connect to the messaging system. It takes a third-party perspective, describing message exchanges in terms of “roles” assumed at runtime by the businesses engaged in a conversation. For carrying on a conversation, each party to a conversation holds its own CP, separately maintains its own internal record of the conversation’s “current state”, and uses the CP to update that state whenever it sends or receives a message. Realizing a conversation model, therefore, requires explicit specification of conversation policies and the ability to transform the conversation rules embedded in a CP into messages that can facilitate the actual conversation, a discourse, among interacting web services.

2.3. Models of Discourse

The interactions between Web Services can be seen as a task-oriented discourse, where the services work in concert to reach a goal. A key approach to task oriented discourse is the plan recognition model [Litman 1987]. In the plan recognition model a task is completed based on domain plan, which defines the topic of conversations and corresponds to a particular way that an utterance can relate to domain plan. The basic unit of discourse formation is the discourse constituent unit [Polanyi 1988]. Elemental units of a “discourse constituent unit” are clauses and discourse operators, which in a conversation can be likened to speech acts [Polanyi 1988]. Clauses consist of one or more words joined together in a syntactically legal manner to make an utterance. A speech-act based approach is appropriate for specifying these utterances because each speech-act consists of an illocutionary force for which expected responses can be specified. They represent the force of the discourse constituents. Discourse operators give information about the state of the discourse and the relation between the discourse entities. Formal specification for the discourse must, therefore, have the qualities of discourse constituent unit. In CS-WS, the conversation policies specified with say, cpXML, hold the plans of discourse between Web Services. Interaction between services takes place in the sequence plan in the cpXML. Each conversation made between services will be based on cpXML. A formal specification of messages must, therefore, underlie the realization of conversation policies. An explicit formal discourse structure is important for realizing the goals of task. In order to communicate efficiently with other services, a formal specification is required for sending and receiving messages and for handling the conversations.

3. Generating Messages to Facilitating Conversations among Web Services

We use a running example to describe our approach. In this example, a customer sends request for loan to a bank; the request is processed by bank; and customer gets to know from bank whether loan was approved. In this example there are three Web Services, named to indicate the roles: Customer, Loan

Approver and Loan Application Processor. The sequence starts with the Customer sending the request for loan. The Loan Approver receives the request and invokes Loan Application Processor service sending it a request to process the loan application. After processing the request, the Loan Application Processor sends approval info to Loan Approver, which sends the reply to Customer. Figure 3 shows a graphical representation of BPEL4WS for loan application process example. BPEL4WS for loan application process is given in Appendix A. Three separate conversations must be facilitated in this process. Figure 4(a) and 4(b) show two of these peer-to-peer conversations with CS-WS: a) between Customer and Loan Approver, and b) Loan Approver and Loan Application Processor. Figure 5 shows the conversation policy for Customer. It is specified as a cpXML for the Customer web service. The cpXML for customer is provided in Appendix B. The generated messages use FLBC syntax and some sample FLBC messages are provided in Appendix C. Formal Language for Business Communication (FLBC) [Moore 2001] is based on Bach and Harnish's version of speech acts [Bach 1979] for specifying messages between services because it has the qualities of discourse constituent unit, and it is in XML format. This formal description defines the interchange vocabulary for the messages that services send, receive and process. FLBC has a small number of general message types, which are formally defined by their intended effects on the recipient.

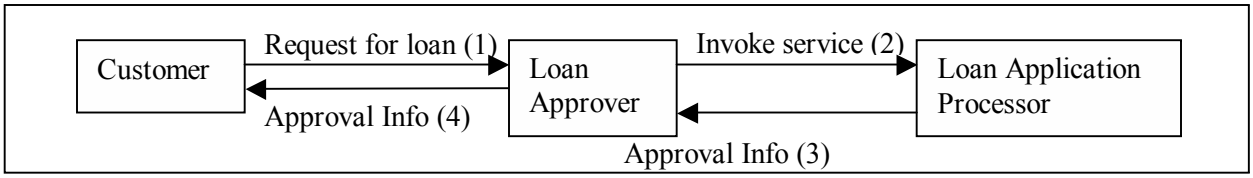

Figure 3: Graphical representation of BPEL4WS for loan application process

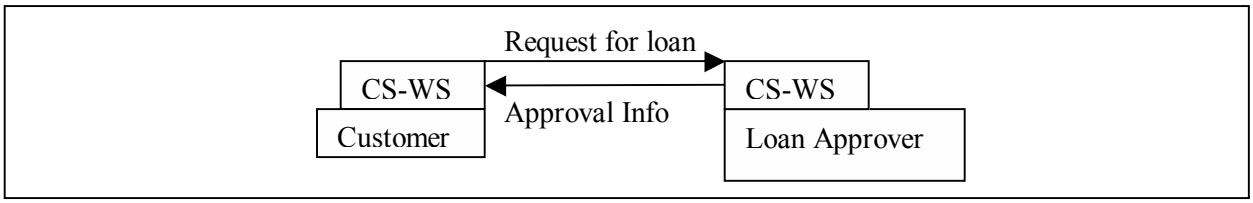

Figure 4 (a): Conversation between Customer and Loan Approver

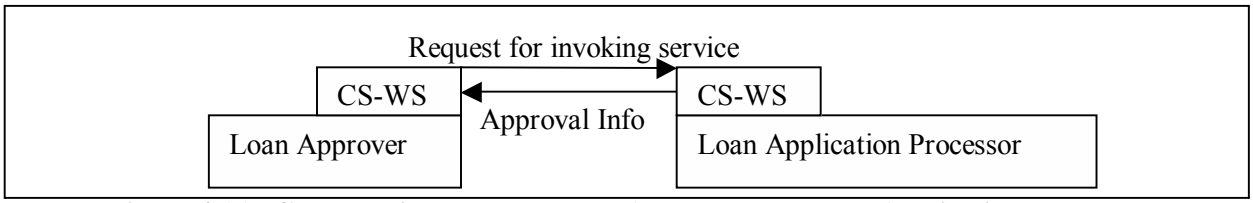

Figure 4 (b): Conversation between Loan Approver and Loan Application Processor

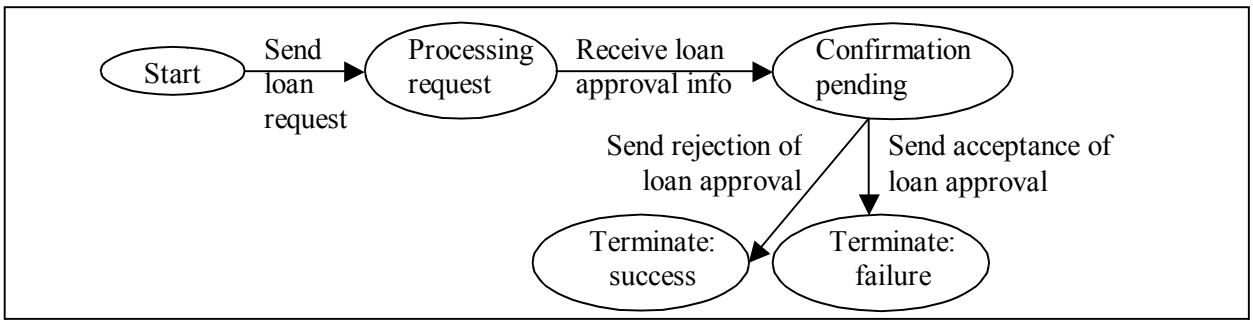

Figure 5: Conversation policy for Customer – loan application process

3.1.Generating FLBC message from cpXML

Figure 6 shows the proposed architecture for generating FLBC message from cpXML. The message generator must work with the CS-WS and the local web service. It uses current state information and message content as inputs and uses cpXML to generate the message. The CS-WS invokes the message generation process and sends the message to other web services. The algorithm for generating FLBC message from cpXML appears following figure 6.

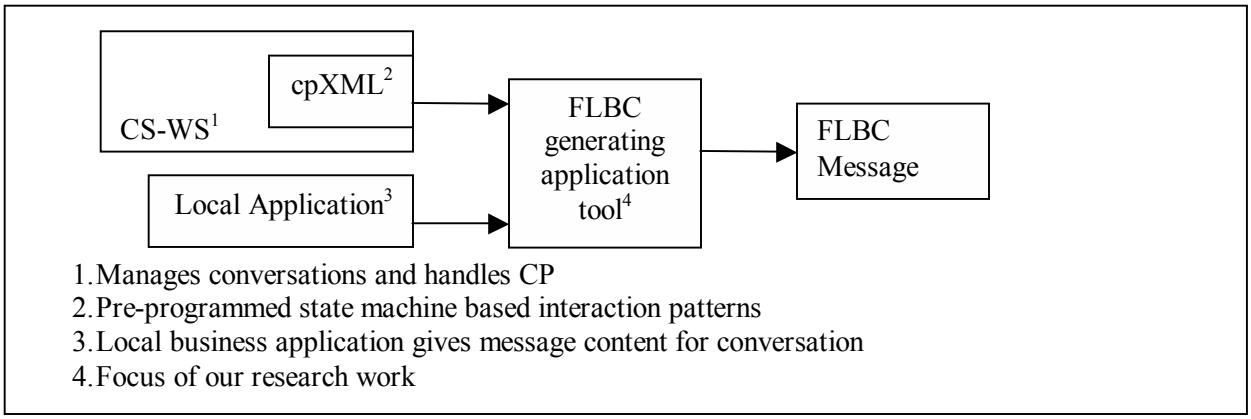

Figure 6: Generation of FLBC message

Algorithm for Generating Messages from Conversation Policies

```
BEGIN (cpXML, currentState, convStack, messageContent, respTo, interRupt)
Open the conversation policy cpXML
        Find stateID with currentState
        Set msgID to stateID
        Set senderName to sender tag
        Set hearerName to other role partner name from the role tag
        Set illForce to schema tag
        If initialstate tag = currentState
                Then Set newConv to "Yes"
        Else Set newConv to "No"
Create new FLBC message file
        Create flbcmsg tag with msgID attribute as msgID
        Create simpleAct tag
                Set speaker attribute to speakerName
                Set hearer attribute to hearerName
        Create illocAct tag with force attribute as illForce
        Create predSt tag with language attribute as value of encoding tag
                Insert value of messageContent agrument
        Create context tag with newConv attribute as newConv
        Create convStack with value as value of convStack argument
        Create sendingMachine with value as service name
        If value of respTo argument isnot null
                then create respondingTo with value as respTo
        If value of interRupt argument is not null
                then create interruptTo with value as interRupt
END
```

4. Discussion and Conclusion

It is envisioned that important productivity gains will come from fully automated machine-to-machine communications. To realize this vision, automated conversation support is a necessary component. Standard messages that can be generated from the conversation policies are an important prerequisite for this component. Conversation support for Web Services, therefore, fills in the important area of handling conversations between Web Services. We have described how an FLBC message can be generated from a specified conversation policy. We are currently implementing the approach proposed in this paper.

References

1. Appendix A-C. http://www.Karthikeyan.Umapathy.com/wits2003/witslinks.xml
2. Bach, Kent; and Harnish, Robert (1979). Linguistic Communication and Speech Acts. MIT Press, Cambridge, MA.
3. BPEL4WS (2003). http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
4. Burg, Bernard (2001) "*Agents in the World of Active Web-Services*". Digital Cities: pg. 343-356.
5. CS-WS (2003). http://www.research.ibm.com/convsupport/faq.html
6. Gille, Marc (2002). "*Workflow Application Architectures: Classification and Characteristics of Workflow-based Information Systems*". Workflow Handbook 2002: p. 39-50.
7. Hanson, James E.; Nandi, Prabir; and Kumaran, Santhosh (2002a) "*Conversation Support for Business Process Integration*". In Proc. of the IEEE International Enterprise Distributed Object Computing Conference (EDOC): pg. 65-74.
8. Hanson, James E.; Nandi, Prabir; and Levine, David W. (2002b) "*Conversation-Enabled Web Services for Agents and E-Business*". In Proc. of the International Conference on Internet Computing (IC-02): pg. 791-796.
9. Litman, Diane J.; and Allen, James F. (1987). "*A Plan Recognition Model for Subdialogues in Conversations*". Cognitive Science (11): pg.: 163-200.
10. Moore, Scott A. (1999) "*On Conversation Policies and the Need for Exceptions*". Autonomous Agents workshop on specifying and implementing conversation policies.
11. Moore, Scott A. (2001) "*A Foundation for Flexible Automated Electronic Communication*". Information Systems Research. 12(1): pg. 34-62.
12. Peltz, Chris (2003) "*Web Services Orchestration- a Review of Emerging Technologies, Tools, and Standards*". Hewlett Packard Labs Technical Paper.
13. Polanyi, Livia; (1988). "*A Formal Model of the Structure of Discourse*". Journal of Pragmatics (12): pg.: 601-638.
14. Seth, Monaj (2003). http://www.developer.com/services/article.php/1497981
15. Sleeper, Brent; and Robins, Bill (2001) "*Defining Web Services*". The Stencil Group.
16. WSCI (2002). http://www.w3.org/TR/wsci/