An infrastructure to support choreographies in interorganizational business processes

Alan Massaru Nakai
IC - Institute of Computing
UNICAMP - University of Campinas
13083-970, Campinas, SP, Brazil
alan.nakai@ic.unicamp.br

Edmundo Madeira
IC - Institute of Computing
UNICAMP - University of Campinas
13083-970, Campinas, SP, Brazil
edmundo@ic.unicamp.br

ABSTRACT

The main attractiveness of Web services is their capacity to provide interoperability among heterogeneous distributed systems. Increasingly, companies and organizations have adopted Web services as a way to interoperate with their business partners. In such a scenario, Web services choreography can be applied in the specification of interorganizational business processes. However, the dynamic nature of business partnerships requires mechanisms for agile designing and deploying of choreographies. In this paper, we present an infrastructure that aims to address the above concern. Our approach, which is based on WS-CDL, BPEL, and UDDI standards, aims to reach flexibility by providing mechanisms for sharing, finding and executing choreographies in a friendly manner for the user. We also present a prototype implementation and its application in a supply chain integration system.

Keywords

Web Services Choreographies, Interorganizational Business Processes

1. INTRODUCTION

Web services are applications that are published, located and invoked over the Web, using XML standards and Internet protocols. The major attractiveness of this technology is its capacity to provide interoperability among heterogeneous distributed systems. Increasingly, companies and organizations have adopted Web services to interoperate with their business partners. In this scenario, services are building blocks that are used for the creation of interorganizational business processes.

Web services-based business processes can be specified as Web services choreographies, which are conversations among multiple services that interact in a collaborative fashion to solve a specific problem. We envision three central issues related to this approach. First, the business environment can be very dynamic; factors such as consumer demand and supplier capacity can lead to changes in partnerships. In consequence, the mechanisms for specification of processes must be flexible enough to accomplish these changes. Second, choreography representations are not directly executable. They need to be mapped to other representations to be executed by specific infrastructures. Finally, partners may not be known at specification time; it can be necessary to discover partners at execution time. In this paper, we present an infrastructure that aims to address the above questions.

In our infrastructure, interorganizational business processes are represented by WS-CDL (Web Services Choreography Description Language) [10], which are performed by a set of coordination managers that execute BPEL4WS (Business Process Execution Language for Web Services) [1] compositions. BPEL is used to represent the specific behavior of one choreography partner and to integrate its public behavior with its internal logic. The infrastructure comprises a WS-CDL to BPEL translator that makes the generation of BPEL code that is executed by coordination managers faster and reduces the possibility of errors and inconsistencies in this task. The specification and deployment of choreographies are facilitated by a coordination management mechanism that spares choreography designers from details of partner binding and context control. The infrastructure allows automatic discovery of partners combining the *roles* abstraction provided by WS-CDL and the categorization mechanism of UDDI (Universal Description, Discovery and Integration) [9]. Choreography descriptions and their related roles are identified by URIs, which are used for categorizing partners in a UDDI registry. This idea, although simple, allows automatic discovery of partners that play specific roles using standard UDDI.

The main contribution of our work is an infrastructure that allows users to quickly find a choreography that matches their necessities, and deploy and execute their portion of the process, despite knowing or not their partners.

This paper is organized as follows. Section 2 presents some related works. Section 3 describes the proposed infrastructure. Section 4 addresses some issues of a prototype implementation and the use of our infrastructure in a supply chain integration system. Finally, in Section 5, we present conclusions and future works.

2. RELATED WORK

Examples of work related to choreography and composition mapping are [8] and [5]. Medling and Hafner [8] present a straight derivation of WS-CDL choreographies to BPEL processes. They show the relationship among BPEL and WS-CDL elements and argue about the derivation limitations. Diaz et al. [5] proposes a mechanism for automatic generation of BPEL skeletons from WS-CDL choreographies. This mechanism uses timed automatas as an intermediary model. According to the authors, the use of timed automatas enables verification and validation of the generated processes.

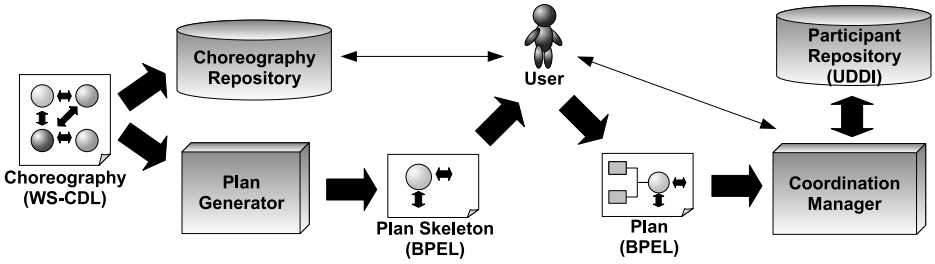

Figure 1: Infrastructure overview.

Chung et al. [4] present the WSCPC (Web Service oriented Collaborative Product Commerce), an architecture for business process management based on Web services orchestrations. This architecture represents business processes using a specific grammatic. The grammatic allows to specify task's dependences, inputs, and outputs. The architecture provides mechanisms to discover services, bind them to tasks, and coordinate them.

Caituiro-Monge and Rodriguez-Martinez[3] and Jung et al. [6] present solutions based on Web services choreography for E-Government and business processes, respectively. Caituiro-Monge [3] introduces a framework for Web services collaboration in E-Government. This framework includes control data in an XML document, which is attached to service requests and partial results. These control data indicate the next Web service to be invoked, the destination of partial data and the way that partial data should be processed. Jung [6] proposes a methodology for business process choreography that incorporates existing workflows into interorganizational business processes. This methodology is based on an architecture that uses an interface protocol which allows interoperability between business partner's internal workflows and external business logic.

3. INFRASTRUCTURE

This section presents the proposed infrastructure. Section 3.1 introduces the choreography representations adopted by our infrastructure. Section 3.2 shows an overview of the infrastructure. Sections 3.3 addresses the mechanism for resolving partners and finally, in Section 3.4 we present the coordination management provided by the infrastructure.

3.1 Choreography Representations

Our infrastructure is based on two levels of choreography representations:

- Global view: it is a document that defines the role of each business partner in the context of a business process. It describes, in a global point of view, the interactions among partners, the conditions for interactions to happen and the set of data exchanged during the collaboration. We have adopted WS-CDL for representing global views. Each global view is assumed to be uniquely identified by a URI (*choreographyID*). In the same way, each role that is defined in the global view is uniquely identified by a URI (roleID) in the context of that global view;
- Coordination plans: Each coordination plan describes the individual behavior of one of the roles defined in a global view. A coordination plan is composed of a set of instructions that trigger operation invocations from other partners, provide operations to other partners and control the flow of actions of the related partner. Besides, the coordination plan integrates the external logic defined by a global view with the internal logic of an individual partner. We have adopted BPEL for representing coordination plans.

The rules and constraints defined in a global view enable an organization to design a coordination plan that reflects its behavior in the correspondent business process. A global view ensures that coordination plans for distinct roles, generated from the same global view, are interoperable. However, global views do not define a partner's specific internal logic. So, an organization that wants to play a role in a business process described by a global view must generate a coordination plan skeleton from that global view and then customize it, including organization's internal logic.

3.2 Overview

Our infrastructure is showed in Figure 1. It is composed of four elements: the *choreography repository*, the *plan generator*, the *coordination manager*, and the *participant repository*.

The choreography repository stores and shares choreography descriptions. It allows users to publish and discover choreography descriptions that fulfills their business requirements. Any organization that wants to participate in a choreography can access a choreography repository to obtain the corresponding description. Such a description is used as the input of a plan generator that creates a coordination plan skeleton. If required, a programmer can customize the skeleton including the organization's internal logic. Each generated plan implements the logic needed for playing a role specified by the global view.

Since the coordination plan is generated, it can be interpreted and executed by a coordination manager. The coordination manager is also responsible for performing the binding of choreography partners. The binding of a partner encompasses the use of a UDDI-based participant repository. The participant repository allows coordination managers (i) to discover a coordination manager that is able to perform a determined choreography role or (ii) to resolve the endpoint reference of a coordination manager that is already

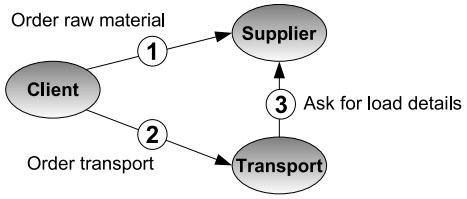

Figure 2: Example of a simplified choreography for raw material supplying.

performing such a role.

To illustrate the use of our infrastructure, consider a simple scenario: a manufactory M needs to implant a business process to order raw material. Therefore, M searches in a choreography repository for a global view that accomplishes its requirements. Assume that M chose the choreography described in Figure 2.

The choreography example defines three roles: *client*, *supplier*, and *transport*. The client partner invokes a specific Web service operation to order raw material from *supplier*. Next, *client* orders transport service from *transport*. To perform its portion of the process, *transport* needs to ask *supplier* for load details. From the choreography global view, M generates the coordination plan (*plan-client*) for role *client* and deploys this plan in its coordination manager.

To execute *plan-client*, M's coordination manager interacts with a participant repository to discover partners that are able to execute coordination plans relative to roles *supplier* and *transport* – in fact, partner binding does not need necessarily to be done through a participant repository; we detail binding issues in later section. After having discovered appropriate partners (consider S and T for roles *supplier* and *transport*, respectively), M's coordination manager can invoke their operations, according to constraints defined by its plan. Partner T's coordination manager, in its turn, needs to interact with the participant repository to obtain the endpoint reference of the specific partner that is playing the *supplier* role in the current instance of the choreography – in the case, S. Figure 3 shows the choreography execution in a high level.

3.3 Resolving Partners

Our infrastructure allows a choreography participant to resolve partners at execution time with small effort of choreography designers. Global views and coordination plans can be designed in a partner role level, in spite of who will play each role. The logic for resolving partner's endpoint references is implemented by the coordination manager and is almost transparent for designers. The coordination manager can obtain a partner endpoint reference by three ways:

- Automatic partner discovery: the coordination manager searches for partners in a participant repository using the identifier of the role (*choreographyId* + *roleId*) the partner must be able to play (e.g. interactions involving partner M in Figure 3). The participant repository returns a list of possible partners and the coordination manager chooses one by consulting a human user or by using a preestablished criterion. Automatic partner discovery should be used when, in the choreography execution, a partner is not previously known and should be initiated by the one that is performing the search.

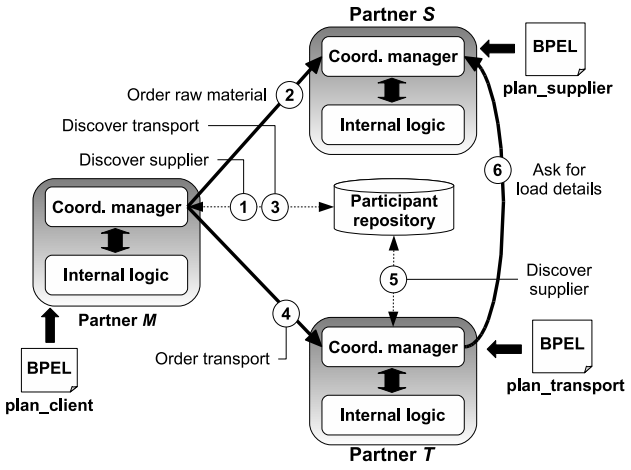

Figure 3: Example execution.

- Automatic instance discovery: the coordination manager searches in a participant repository for the partner that is already playing a role in a specific choreography instance (e.g. interaction among partners T and S in Figure 3). The key used for searching is the combination of the identifiers of the role and the choreography instance. Automatic instance discovery should be used when, in the choreography execution, a partner is not previously known and is already initiated.

- Static addressing: the endpoint reference of the partner is defined at deployment time of the coordination plan. Static addressing should be used when the partner is known before the choreography execution.

The coordination manager maintains a configuration file for each coordination plan that it is able to execute. This configuration file, which is created by the plan designer, defines the form for resolving the endpoint reference for each partner.

We have specified the participant repository using UDDI. Each business partner registers a *uddi:businessEntity*[1] element in the UDDI registry. This element presents partner's information, such as description and contacts. UDDI categorization mechanism is used to associate partners and roles. *Uddi:businessEntity* elements are categorized according to the identifier of the roles that the partner can play. Automatic partner discovery is enabled through the *find_business* operation, provided by the UDDI inquiry API, using the role identifier as category.

The *uddi:businessService* is used to represent an instance of the role that is executing and holds the partner endpoint reference. When a coordination manager starts a coordination

[1]We use the prefix *uddi* to denote UDDI elements.

plan, a new *uddi:businessService* is created in order to represent the instance of that plan. The *uddi:businessService* is categorized according to the identifier of the role it is related to and the identifier of the choreography instance. Automatic instance discovery is enabled by *find_service* and *find_binding* operations, provided by the UDDI inquiry API, using identifiers of instance and role as categories.

3.4 Coordination Management

An important issue when executing choreographies is to preserve the context of choreography instances to ensure that messages arrive to the right instances of coordination plans. Coordination managers provide a coordination management abstraction that controls context, in a level that is hidden from designers. It means that choreography designers can design global views and coordination plans without concerning about context control.

Interactions between two coordination managers are preceded by a connection phase. In this phase, coordination managers agree on the choreography to be followed, the role to be played by each one, and the context identifier that distinguishes the choreography instance. The connection phase is initiated when a coordination plan instance issues a message that must be sent to a disconnected partner. To connect to a partner, the coordination manager resolves the partner endpoint reference (as shown in Section 3.3), sends it a connection proposal and waits for a connection confirmation. If the connection is refused, the coordination module throws an exception message.

When the Coordination Manager receives a connection proposal, it must be validated. A connection proposal may be invalidated due to two reasons. First, the coordination manager may be unable to execute the appropriate coordination plan in order to play the destination role. Second, an identical connection may be already active. In the future, we intend to add authentication, authorization, and security mechanisms in the connection validation.

If the connection proposal is validated, the coordination manager checks if the appropriate coordination plan is already in execution. If the Coordination Plan is not executing, the coordination manager starts it and registers the plan instance in the participant repository. Next, a confirmation message is sent back to the coordination manager that has requested the connection.

All application messages – those exchanged by coordination plan instances – carry, in their headers, data used in context control and message delivery. Such data includes identifiers for: choreography (*ChoreographyID*), choreography instance (CIID), sender partner (SRID - *Sender roleID*), and destination partner (DRID - *Destination roleID*). These data uniquely identify a connection between two coordination managers. Through them, the sender coordination manager can retrieve the endpoint reference of the destination one from a table of connections. In other side, the destination coordination manager can deliver the message to the right coordination plan instance.

Figure 3.4 shows the coordination manager architecture. It is composed of two main components: the execution engine

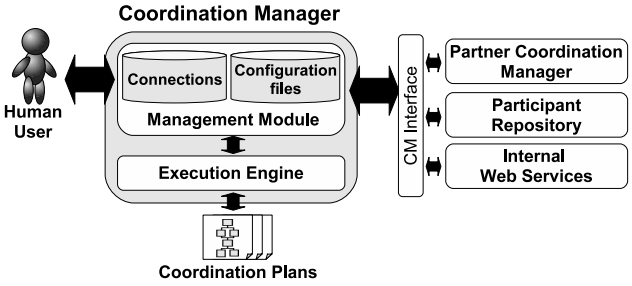

Figure 4: Coordination Manager Interface

.

and the management module. The execution engine interprets and executes coordination plans. It issues messages to the management module and processes messages that are delivered by it, according to the plan flow control. The execution engine can execute more than one instance of a coordination plan simultaneously. Moreover, it can execute distinct coordination plans simultaneously.

The management module is responsible for the interaction with other coordination managers, participant repositories, and Web services that represent the internal logic of the organization. The management module implements the logic for resolving partner endpoint references, managing connections, forwarding messages that are generated by the BPEL engine to destination partners, and delivering messages that are received from partners.

In addition, the management module provides an interface for human user interaction. This interface is useful when, during a coordination plan execution, some decisions must be taken by human users. For instance, an exception message received as response to a Web service request can trigger an alarm. This alarm could ask a human actor whether the coordination plan execution should continue or stop.

The user interface also allows human users to request control operations. We have specified two control operations: *getStatus*, which returns the execution status of a coordination plan to the user, and *abort*, which forces coordination plan termination.

4. PROTOTYPE IMPLEMENTATION

In order to validate our infrastructure, we have implemented a prototype. In the following, we present some issues about our implementation experience:

Coordination manager: we have used a conventional BPEL engine[2] as the execution engine. For this reason, it was necessary to specify a basic structure that all coordination plans must follow to provide a specific interface for the management module. The normal flow of the coordination plan is initiated when the management module invokes an *ExecutePlan* operation. After *ExecutePlan* response, the specific logic of a choreography role is placed. This section of code may be customized by a coordination plan designer.

[2]ActiveBpel, available in http://www.activebpel.org/.

Table 1: Main rules for WS-CDL/BPEL elements mapping.

WS-CDL Element	BPEL Mapping
cdl:variable	bpel:variable
cdl:getVariable	bpel:getVariableData
cdl:globalizedTrigger	Subexpression related to the specific role
cdl:sequence	bpel:sequence
cdl:parallel	bpel:flow
cdl:choice	bpel:switch/bpel:cas
cdl:workunit	Combination of bpel:while and bpel:switch/bpel:case
cdl:silentAction	bpel:empty
cdl:assign	bpel:assign
cdl:interaction	Combination of bpel:invoke and bpel:receive

The normal flow ends when the BPEL engine reports termination by a *PlanConclusion* operation request. The control operations (*GetStatus* and *Abort*) are specified in the skeleton as BPEL event handlers. The basic structure also defines a default *partnerLink*[3] element, which represents the link between the execution engine and the management module, and a default *correlationSet*[4], which enables the composition engine to use message context data in the context control. The management module was implemented as a Java servlet to receive and respond SOAP messages over HTTP. SOAP processing was implemented using JAXM (Java API for XML Processing). Data persistence was made by a Postgres database.

Participant repository: we have implemented it as a simple Web service that is able to register and discover endpoint references. We have implemented only the operations that are needed to test coordination manager functionalities.

Plan Generator: we have specified a set of rules for translating WS-CDL global views to BPEL coordination plans. The central idea of this translation is to generate a coordination plan skeleton for each role defined in the global view. The skeleton must follow the basic structure mentioned bellow. After that, for each WS-CDL element that is related to a specific role, a BPEL mapping is added in the correspondent coordination plan. Table 1 summarizes the main rules for mapping WS-CDL elements to BPEL elements.

We have adopted some restrictions when using WS-CDL language in order to adjust it to our needs. For example, in a WS-CDL choreography, a role is specified as a set of behaviors. To simplify the translation, we have limited in one the number of behaviors of a role. This limitation does not affect our model, because a coordination manager can play more than one role. It is equivalent to play a single role with multiple behaviors. Another restriction is the "elimination" of the WS-CDL *channelType* element. This element defines the endpoint of a choreography partner, the context information used in its message exchanges and some channel constraints. In our architecture, endpoint and context information are implicitly dealt by the coordination manager.

[3]PartnerLink: BPEL element that represents a relationship among partners.

[4]CorrelationSet: BPEL element that defines the fields within a message that must be used as the key of context control

Choreography repository: it can be implemented as a Web site with information and links for each choreography description. However, we are working in a mechanism based on semantic information to facilitate to find the choreography description.

4.1 Using in a Supply Chain Integration System

Our infrastructure has been developed inside a larger project that aims to provide an architecture for supply chain integration [2, 7]. This architecture defines a set of components that treat supply chain management issues, such as coordination of activities, contract negotiation, and product traceability.

The coordination manager is the component that is responsible for coordinating supply chain activities. These activities include business activities, such as purchase requests and invoice emissions, and also management activities, such as the initiation of other architecture components. All activities are triggered by Web services invocations, which are specified by coordination plans.

Other components are: the negotiation manager, which is responsible for negotiating business contracts whit partners, the summary manager, which manages the supply chain traceability, and the regulation manager, which aims to assure that supply chain regulations are maintained.

In order to illustrate the use of our infrastructure in the supply chain integration system, we present, in this section, a simple case study based on the example shown in Section 3.2. Let us consider, however, that *client* quotes prices for raw material at two *suppliers* and requests a purchase order from the one with the cheaper offer. Figure 5(a) shows a global view for this scenario, represented in pseudo-WS-CDL notation. Lines 1 to 7 define two *interactions* that specify quote requests from *client* to suppliers. The *workunits* at lines 8 to 13 compare the obtained quotes so that *client* can invoke the *purchaseOrder* operation of the cheaper supplier. Lines 14 and 15 define the invocation of the *transportOrder* operation from *client* to *transport*. Finally, the *workunits* at lines 16 to 23 define the logic needed for *transport* to resolve whether it invokes the *getLoadDetails* operation from *supplier1* or *supplier2*.

Figure 5(b) presents the coordination plan, generated from the global view of Figure 5(a), for the *client* role. In this plan, which is represented in pseudo-BPEL notation, the highlighted code is related to the *client* internal logic. The *switch/case* construction at lines 7 to 11 adds the logic to request the negotiation manager to negotiate a contract with *supplier1*. If the negotiation manager successfully completes the negotiation, *client* proceeds the transaction and invokes the *purchaseOrder* operation from *supplier1*. If the negotiation fails, *client* terminates the plan execution. The *supplier2*'s equivalent code was omitted for simplicity. The *request* at line 14 asks the summary manager to register the completed transaction in order to enable the traceability of the purchased raw material.

5. CONCLUSION

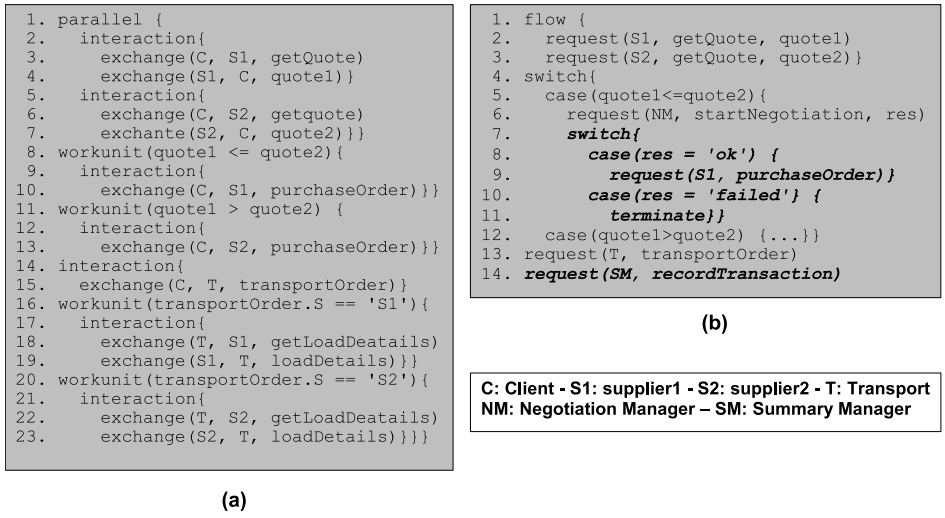

Figure 5: (a) Case study global view (pseudo-WS-CDL). (b) Coordination plan for the *client*.

We believe that an important issue in SOA environments is the flexibility to agilely adapt to business changes. Our solution aims to reach such flexibility by providing mechanisms for sharing and finding choreography descriptions and deploying and executing them with small programming effort. The prototype implementation shows our infrastructure's feasibility.

A contribution of our infrastructure is a mechanism based on standard UDDI that allows automatic partner discovery. This feature allows business processes to be executed by distinct configurations of partners. Another contribution is the separation between business logic and coordination management. Choreography designers do not need to worry about the way context control will be done or which choreography partners are connected to the system. This transparency facilitates the choreography design.

Our infrastructure also facilitates the deployment and execution of choreographies. The plan generator makes faster the generation of coordination plans; besides, it reduces the possibility of errors and inconsistences in this task.

There are some issues that will be addressed as future work: (i) the use of semantics for choreography descriptions searching; (ii) the improvement of the coordination manager architecture to deal with partner authentication, security and authorization issues; and (iii) the provision of fault tolerance.

6. ACKNOWLEDGMENTS

The authors would like to thank CAPES, FAPESP, and CNPq WEBMaps and AgroFlow projects for the financial support.

7. REFERENCES

[1] T. Andrews et al. Business Process Execution Language for Web Services - version 1.1, May 2003.

[2] E. Bacarin, C. Medeiros, and E. Madeira. A Collaborative Model for Agricultural Supply Chains. In R. Meersman and Z. Tari, editors, *CoopIS/DOA/ODBASE 2004, LNCS 3290*, pages 319–336. Springer-Verlag, 2004.

[3] H. Caituiro-Monge and M. Rodríguez-Martinez. Net Traveler: A Framework for Autonomic Web Services Collaboration, Orchestration and Choreography in E-Government Information Systems. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 2–10, Washington, DC, USA, 2004. IEEE Computer Society.

[4] M. J. Chung et al. A framework for collaborative product commerce using web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 52–60, 2004.

[5] G. Diaz, V. V. M. E. Cambronero, J. J. Pardo, and F. Cuartero. Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 186–192, 2006.

[6] J. Jung, W. Hur, S.-H. Kang, and H. Kim. Business Process Choreography for B2B Collaboration. *IEEE Internet Computing*, 8(1):37–45, 2004.

[7] A. A. Kondo et al. Web services-based traceability in food supply chains. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, 2007.

[8] J. Mendling and M. Hafner. From inter-organizational workflows to process execution: Generating BPEL from WS-CDL. *Proceedings of OTM 2005 Workshops. Lecture Notes in Computer Science*, 3762:506–515, 2005.

[9] OASIS. Uddi version 3.0.2, 2004. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

[10] W3C. Web Services Choreography Description Language version 1.0, November 2005.