Generation of choreography skeletons from web service definitions

Annett Laube and Patrick Winkler
Bern University of Applied Science
Devision of Computer Science
Biel/Bienne, Switzerland
Email: annett.laube@bfh.ch, winkp1@bfh.ch

***Abstract*—Modern IT landscapes underlie constant evolution. Modeling activities - as a basis for continuous monitoring and maintenance - stay often behind. Service models describe the way how services interact. We propose reverse engineering techniques to generate choreography skeletons from web service definitions. We describe the necessary transformation to generate a detailed and consistent choreography than can be easily completed and merged with existing choreographies. We discuss the restrictions of the generated WS-CDL skeletons and how they can be overcome.**

***Index Terms*—web service, choreography, service model, reverse engineering**

I. Introduction

IT landscapes in industry or finance are often a result of a long evolution. Despite continuous efforts to keep models of systems and components up-to-date, rarely IT landscape models are consistent and complete.

The *landscape model* (sometime also called system model) consists often of two parts: the physical infrastructure and the service model. The physical infrastructure reflects in great detail components composed in the following groups: computer hardware (e.g. processing power, memory, etc.), computer software (e.g. OS, server applications), and network devices (e.g. links, traffic controllers - hubs/switches/gates/routes).

The *service model* is an abstraction of the set of all services. It defines the way how the services interact by exchanging messages and how more complex services are created by combining services. The terms *orchestration* and *choreography* describe two different aspects of creating business processes from composite web services.

Orchestration refers to an executable business process that can interact with both internal and external web services. Orchestration represents the composition from the viewpoint of the parties involved in this composition.

A *choreography* description concerns the composition of web services seen from a global viewpoint focusing on the common and complementary observable behavior. Choreography is particularly relevant in a setting where there is not a single coordinator. Choreography tracks the message sequences among multiple parties and sources - typically the public message exchanges that occur between web services - rather than a specific business process that a single party executes [1]. Typical examples are a travel agency that offers a broad range of services including air and train travel, bus tickets, hotels, car rental, excursions, insurance etc. or a company that wishes to purchase a fleet of cars from automobile suppliers, which in turn request quotes for specific bill or material items from their component manufacturers [2].

The main use of a choreography description is to precisely define the sequence of interactions between a set of cooperating web services in order to promote a common understanding between participants and to make it as easy as possible to automatically validate conformance, ensure interoperability and increase robustness [2].

We want to automate the generation of service models, more specific of web service choreographies. This reduces the modeling effort for existing service landscapes. Normally, IT service landscapes underlie constant evolution due to newly added or modified business functions. Quite often web services from partners or external service providers have to be integrated into the existing network. Examples are services from B2B partners or external data services, like Dun&Bradstreet to get business information and company assessments from business partners. In this case, our work helps to keep the service models up-to-date and to monitor changes in constantly changing service infrastructures.

We use *reverse engineering* techniques to extract the necessary information from the implemented web services. Main information source are the web service definitions, which provide the documentation for distributed systems and are available to all communication partners. As the information in the web service definitions is insufficient to build a complete service choreography, we concentrate on the generation of consistent skeletons, which then can be enhanced manually or enriched with business process information.

The paper continues with a description of related work. The basis features of WS-CDL and WSDL are addressed in Sections III and IV. In Section V, the transformation from web service descriptions to a choreography model is described. In Section VI, we discuss how the generated choreography skeletons can be completed. Our implementation of the transformation process is described in Section VII. In Section VIII, we conclude this paper and discuss further work.

II. Related Work

The most known languages to specify web services choreographies are Web Service Choreography Interface (WSCI,

[3]), Web Service Choreography Description Language (WS-CDL, [4]), and Ontology Web Language for Services (OWL-S, [5]). All are XML-based and support WSDL [6][7], the well-established standard to describe web services.

WSCI – sometimes considered as predecessor of WS-CDL (the last update was released in 2002) – describes the observable behavior of only one web service including temporal and logical dependencies in the message flow. WS-CDL and OWL-S are more powerful to express the collaboration of 2 parties. WS-CDL describes peer-to-peer collaborations of web services taking part in a choreography. It defines a set of agreements about ordering and constraint rules. The aim of OWL-S is to automate the discovery, invocation, composition, interoperation and monitoring of web services. A detailed comparison of the 3 languages can be found in [8].

More choreography languages, like Let's Dance and BPEL4Chor, are emerged in the last years. Let's Dance [9] is a language for modeling service interactions and their flow dependencies targeting business analysts. It is a language for high-level analysis and design. WS-CDL is a potential implementation language for Let's Dance models. BPEL4Chor [10] is an extension of Business Process Execution Language (BPEL, [11]). It adds participant behavior descriptions, i.e. control flow dependencies, the participant topology and their interconnection using message links and participant groundings, i.e. concrete configurations for data formats and port types to the standard BPEL.

To our knowledge, it is a novel approach to reengineer WSDL files to create a service model. But there are reverse approaches to generate WSDL descriptions (skeletons) from choreography models. In [12], the authors describe an approach to generate the orchestration behavior (BPEL stubs) and the necessary WSDL templates automatically from WS-CDL models. The same functionality is implemented in the visual modeling tool known as pi4SOA [13]. pi4SOA, an open-source implementation that plugs into Eclipse, is one of the few WS-CDL implementations available today.

There are many approaches, complementary to our approach, to recreate the process flow of interacting web services. *Business process mining*, or *process mining* for short, aims at the automatic construction of models explaining the behavior observed in the event log [14]. For example, based on event logs, it is possible to construct a *process model* expressed in terms of a Petri net or Event-driven Process Chain (EPC). Beside the process perspective, process mining can also focuses on the originator field (organizational perspective), to find out, which performers are involved and how are they related. The goal is either to structure the organization by classifying people in terms of roles and organizational units or to show relations between individual performers (e.g., build a social network [15]).

III. Choreography

The goal of specifying web service choreographies is composing peer-to-peer interactions between any kind of services, regardless of the programming language or the environment that hosts the service.

We have chosen to use WS-CDL for our service model, because its tight coupling to WSDL files and the recommendation of the W3C Web Services Choreography Working Group.

A WS-CDL model consists of 3 parts:

- **Collaborating parties:** describing the entities that exchange information, their roles and their relationships,
- **Collaborative behavior**: describing the physical order (message flow) of the information exchange and assigned constraints,
- **Exchanged information:** describing the type of information used in the information exchange.

A. Collaborating parties

Within a choreography, information is always exchanged between participants. A participant – described by a *participant type* – groups all the parts of the collaboration that must be implemented by the same entity. A *role type* enumerates potential behaviors of a participant within an interaction. A *channel type* is a point of collaboration between participants specifying where and how information is exchanged. Finally, a *relationship type* is used to identify the mutual obligations between participants that must be fulfilled to succeed.

B. Collaborative behavior

A *choreography* defines re-usable common rules that govern the ordering of exchanged messages. A choreography contains collections of *activities* that may be performed by one or more participants. There are three types of activities in WS-CDL, namely *control-flow* activities, *WorkUnit* activities and *basic* activities. In the first category, there are three types of activities: *Sequence*, *Parallel*, and *Choice*. These activities enclose a number of sub-activities. A *WorkUnit* activity describes the conditional and, possibly, repeated execution of an activity. The *basic* activities include *Interaction*, *NoAction*, *SilentAction*, *Assign*, and *Perform*. The most important element of WS-CDL is the *Interaction* activity that corresponds to an operation of a web service.

C. Exchanged information

InformationTypes describe the type of *variables*, *tokens* and *messages* used in the choreography. Their description at the package level makes them available to all enclosed activities. They normally refer either WSDL 1.1 message types, WSDL 2.0 schema elements or XML schema elements/types.

IV. WSDL

A web service is described by a web service description (WSDL file). Currently, there are 2 versions of the specification. WSDL 1.1 [6] is the widely accepted standard. Although WSDL 2.0 [7] is recommended by the W3C since June 2007 and promises an easier implementation, its adaption by SOAP servers, vendors and tools is still reluctant.

A WSDL 1.1 description containing six major elements (In this paper, we concentrate on WSDL 1.1, but the same information is also available in WSDL 2.0.):

- *types*, which provides data type definitions used to describe the exchanged messages.
- *message*, which represents an abstract definition of the data being transmitted.
- *portType*, which is a set of abstract *operations*, which refer to input and output messages.
- *binding*, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- *port*, which specifies an address for a binding, thus defining a single communication endpoint.
- *service*, which is used to aggregate a set of related ports.

To generate the choreography skeletons, only the elements *message*, *portType* with *operations* and *service* are used. The name of *service* element is used to name the activities and collaborating parties in the choreography.

V. TRANSFORMATION

Our goal is the generation of a valid web service choreography (*.cdl file) out of one or several web service definitions (WSDL files). In the following, we describe the needed transformations to create the 3 essential parts of a WS choreography.

A. Collaborating parties

The *service* element of a web service description is used to generate the collaborating parties of the choreography. Each service element represents a relationship between service provider and service consumer. The name of the *service* element is used to generate the relationship type and the role types related to web service provider and consumer. In Figure 1, the graphical representation of such a relationship is shown.[1]

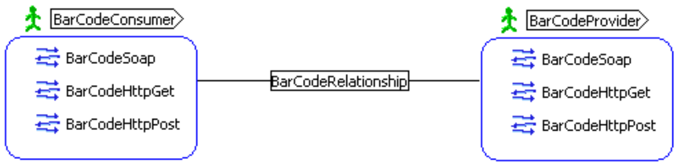

Fig. 1. Graphical Model of a WS-CDL Relationship

The role types enumerate potential observable behaviors a participant can exhibit in order to interact. This behavior corresponds to the *portTypes* (*interfaces* in WSDL 2.0) of the WSDL. In Figure 1, each role type has 3 behaviors assigned.

The optional behavior of the role types in the relationship is not filled during the transformation. In this case, all the behaviors belonging to this role type are identified as the commitment of a participant for this relationship. The right values have to be selected manually in a later stage.

The generated relationship type is assigned to the choreography (see in Figure 2) and bound to all interactions created from the WSDL file (see Section V-B).

[1]In this and the following examples, we used a public web service for bar code generation available with its WSDL under http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=40.

```
<package>
  <roleType name="BarCodeProvider">
    <behavior interface="BarCodeSoap" name="BarCodeSoap"/>
    <behavior interface="BarCodeHttpGet"
       name="BarCodeHttpGet"/>
    <behavior interface="BarCodeHttpPost"
       name="BarCodeHttpPost"/>
  </roleType>
  <roleType name="BarCodeConsumer">
    <behavior interface="BarCodeSoap" name="BarCodeSoap"/>
    <behavior interface="BarCodeHttpGet"
       name="BarCodeHttpGet"/>
    <behavior interface="BarCodeHttpPost"
       name="BarCodeHttpPost"/>
  </roleType>
  <relationshipType name="BarCodeRelationship">
    <roleType typeRef="BarCodeProvider"/>
    <roleType typeRef="BarCodeConsumer"/>
  </relationshipType>
  <choreography name="BarCode" root="true">
    <relationship type="GlobalWeatherRelationship"/>
    ...
  </choreography>
</package>
```

Fig. 2. WS-CDL Relationship

Participant types are not generated. The participants are the logical entities or organizations implementing or using the web services. The necessary information is not available in the WSDLs of the web services and can be added later.

```
<informationType element="AnyType" name="BarCodeRef"/>
<token informationType="BarCodeRef" name="BarCodeRef"/>
...
<channelType action="request-respond"
     name="GenerateBarCodeChannel11"
     usage="distinct">
  <roleType typeRef="BarCodeProvider"/>
  <reference>
    <token name="BarCodeRef"/>
  </reference>
</channelType>
```

Fig. 3. WS-CDL ChannelType and related token

A channel type realizes a point of collaboration between participant types by specifying where and how information is exchanged. All our channel types have mostly the action type `request-respond`. In the rare cases, that a web service operation has no parameters or does not return anything (that means there is no input or output message assigned to the operation) the action type *respond* rsp. *request* is assigned. A channel type is named (the name is generated from the *wsdl:operation*) and then related to the role of the web service provider. In the case of several behaviors (corresponding to the *portTypes* in the WSDL), we generate equally channel types. The first of these is related to the interaction via a channel variable. A channel type gets a reference assigned to convey the information needed to contact the receiver of the message. This *reference token* is associated with an information type. As the reference information belongs to the business process, we can only generate the tokens and a dummy information type (see in Figure 3).

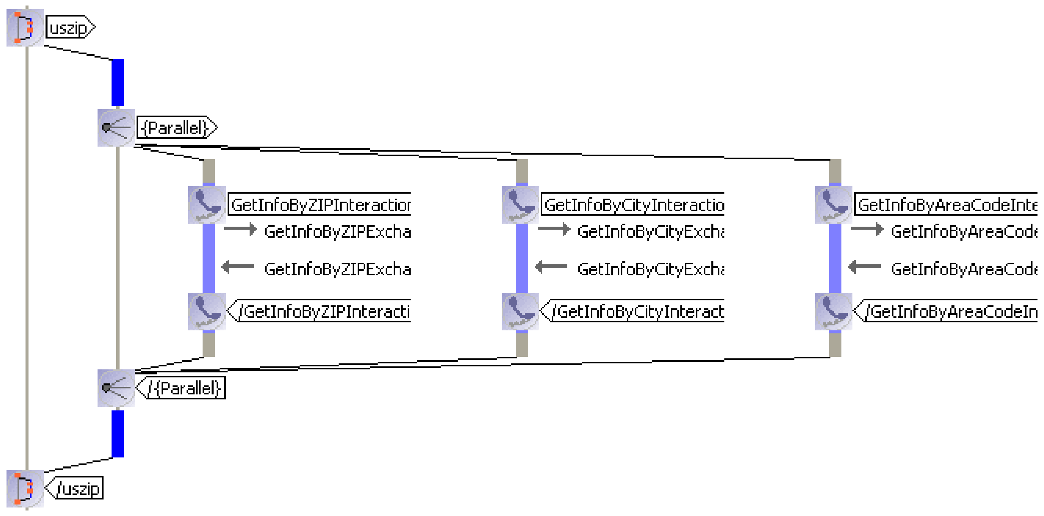

Fig. 4. Graphical Model of Parallel Activities

B. Collaborative behavior

The collaboration behavior of two partners is described by *activities* with an ordering structure. The web service definition contains no information about the order in which the different operations are called, therefore we assume an unrestricted *parallel* activity, in which one or more activities can be executed in any order or at the same time (see Figure 4[2]). By nature, all operations of a web service can be called at any time and in any order. Ordering restrictions are only given by the using business process. Our skeletons could be enriched later with the real ordering structure, either manually or automatically by integrating the information from, e.g., a business process model.

The *basic activities* generated from the WSDL files are *interaction* activities. Each web service operation corresponds to one *interaction* (see in Figure 5 a generated example interaction). The name of the web service operation is used to generate a name for the interaction and the `operation` attribute.

An interaction activity description has 3 main parts corresponding (i) to the participants involved, (ii) to the information being exchanged, and (iii) to the channel for exchanging the information.

The information about the involved participants is contained in the element *participate* and refers to the role types and relationship types described in Section V-A. The attribute `fromRoleTypeRef` refers the web service consumer role, the attribute `toRoleTypeRef` to the web service provider role, and the attribute `relationshipType` to the relationship between the two.

The exchanged information is described in the *message* part of a WSDL. The operation *input* and *output* elements connect the related messages to a WSDL operation. The information from the *message* element is transformed to the

[2]To illustrate parallel activities, we have chosen a weather web service available under http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=48

```
<informationType type="GenerateBarCodeType"
name="GenerateBarCode"/>
<informationType type="GenerateBarCodeResponseType"'
name="GenerateBarCodeResponse"/>
...
<choreography name="BarCode" root="true">
...
  <parallel>
    <interaction
        channelVariable="GenerateBarCodeChannelVariable1"
        name="GenerateBarCodeInteraction"
        operation="ReceiveGenerateBarCode">
      <participate fromRoleTypeRef="BarCodeConsumer"
          relationshipType="BarCodeRelationship"
          toRoleTypeRef="BarCodeProvider"/>
      <exchange action="request"
          name="GenerateBarCodeExchange1"
          informationType="GenerateBarCode">
        <send/>
        <receive/>
      </exchange>
      <exchange action="respond"
          name="GenerateBarCodeExchange2"
          informationType="GenerateBarCodeResponse">
        <send/>
        <receive/>
      </exchange>
    </interaction>
    ...
  </parallel>
</choreography
```

Fig. 5. Generated WS-CDL Interaction

exchange element in the WS-CDL interaction. Depending on the operation type the `action` attribute of the exchange is generated differently: input messages → `action=request` and output message → `action=respond`.

The attribute `informationType` of the *exchange* element refers to the information type used for the exchanged information. All information types are defined at the package level of the choreography. The elements *send* and *receive*, which contain application-dependent or state information are generated without attributes.

The channels used during the interactions are also derived from the web service operations. The associated channel types are defined on package level and only referred via channel variable in the interaction (attribute `channelVariable`).

C. Exchanged information

Information types are mainly used in the *exchange* element of the interaction to describe the type of exchanged information. WS-CDL does not allow the construction of complex data types like possible in the *types* element of the WSDL. Therefore we have to generate new information types from the data type assigned to the WSDL 1.1 message parts.

The information types contain also the generated information type for the *reference tokens* (see Figure 3).

The complete mapping of web service definitions to a WS-CDL choreography is shown in Table I. Although the described transformation uses a single WSDL, the concept can equally applied to several files. Naming conflicts are anticipated by applying different namespaces for WSDL specific and generated elements.

VI. Skeleton completion

The following steps are necessary to complete the generated WS-CDL skeletons and to merge them into existing choreographies:

1) Create the participant types and assign the generated role types: Typically a participant type groups several roles. In a chain of interacting services, a participant can be the consumer of one service and the provider of another. Process mining techniques could be used to identify the roles that belong to one participant type.
2) Select the implemented behaviors in role types of the service consumer: The behaviors generated from the *portTypes* describe the different possibilities to communicate. But a specific service consumer could decide to use only a certain subset. This information belongs to the web service consumer's client application.
3) Verify the behaviors of the relationship types: Per default, all generated behaviors are committed from both sides. But for a specific combination of service consumer and service provider only a subset could be used (in accordance with the selection in 2).
4) Remove or flag unused channel types: A channel type describes the communication channel for each interaction in accordance with a selected behavior. If a web service consumer uses only a subset of the communication channels, some of the generated channel types become obsolete and can be removed.
5) Select the correct channel for each interaction: Each interaction has exactly one channel type assigned. Per default, we select always the first generated channel type. In accordance to the selected behaviors for the relationship type, the right channel type has to be selected. If the interaction can be executed on several channels the interaction has to be duplicated.
6) Fill additional information: Information from the business process and description of all components can be added.
7) Merge the choreography: The completed skeleton can now be merged with existing choreographies. This is a manual step that is not supported by the pi4SOA tool [13].
8) Establish the flow: The last step is to establish the correct ordering structure of the activities from the merged choreographies. The generated parallel activities have now to be brought in the right sequential order and to be integrated in the complex flow of activities. As information source serves mainly the knowledge about the business process. Existing workflow descriptions of the business process or process models constructed with process mining techniques could enrich the choreography.

VII. Implementation

Our transformation process consists of 3 automated steps (see Figure 6) complemented by manual activities to add additional information and to merge the skeletons with existing choreographies (see Section VI).

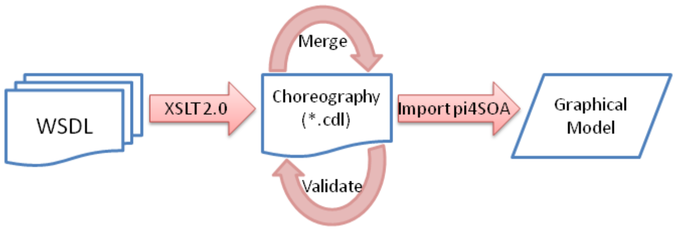

Fig. 6. Transformation flow

The transformation process starts with the selection of one or several WSDL files. A XSLT transformation transforms the input into a *.cdl file containing the choreography skeleton in WS-CDL. We use a XSLT 2.0 engine. The *.cdl file can now be manually completed and merged with existing choreographies. After this, a validation against the XML schema provided by [4] verifies the correctness of the manual editing. In the last automated step, the *.cdl file is imported into the Eclipse Plug-in pi4SOA [13]. During the import, the *.cdl file is semantically validated and transformed in a graphical model (stored as *.cdm file) that can be visually modified.

VIII. Conclusion and Future Work

We presented an approach to generate WS-CDL skeletons from web service definitions. The result is promising; we could generate complete and consistent choreographies that can be easily completed manually with the support of a graphical tool. Our approach of reengineering web service definitions facilitates the modeling process (the modeling time is reduced from several hours to a couple of minutes). It generates skeletons that describe the web service operations detailed as interactions including the cumbersome modeling and referencing of communication channels, behaviors, relationships, etc.

WSDL		WS-CDL	
Element	*Attribute*	*Element*	*Attribute*
message → part	element	informationType	name
(name="parameters")	element+"Type"	informationType	type
message → part	element	interaction → exchange	informationType
portType	name	roleType → behavior	name
	name	roleType → behavior	interface
portType → operation	name	channelType	name
	"request-respond"/"request"/"respond"	channelType	action
	"distinct"	channelType	usage
portType → operation	name+"Interaction"	interaction	name
	"Receive"+name	interaction	operation
	name+"ChannelVariable"	interaction	channelVariable
portType → operation → input	"request"	interaction → exchange	action
portType → operation	name+"Exchange[n]"		name
portType → operation → output	"respond"	interaction → exchange	action
portType → operation	name+"Exchange[n]"		name
portType → operation → fault	"respond"	interaction → exchange	action
	name		faultname
portType → operation	name+"Fault"		name
service	name+"Provider"	roleType	name
	name+"Consumer"	roleType	name
	name+"Relationship"	relationshipType	name
	name+"Provider"	relationshipType → roleType	typeRef
	name+"Consumer"	relationshipType → roleType	typeRef
service	name+"Relationship"	choreographie → relationship	type
service	name+"Ref"	informationType	name
	"AnyType"	informationType	element
	name+"Ref"	token	name
	name+"Ref"	token	informationType

TABLE I
WSDL TO WS-CDL MAPPING

We plan to further reduce manual modeling efforts by automated enrichments of the choreography model from existing workflow descriptions, like BPEL, or process models constructed by process mining. In [12], orchestration behavior was generated from a choreography. We will also try to reverse this process.

We consider our work as a first step in the direction of automated service model generation as a basis for constant monitoring of steadily evolving service landscapes. So far, we concentrated on the functional feature of web services. In the future, we want also consider non-functional aspects, like security and dependability. This will require extensions to the choreography languages, like WS-CDL.

ACKNOWLEDGMENT

This work was partially funded by the European Commission under the Seventh Framework Project "PoSecCo" (IST 257129).

REFERENCES

[1] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[2] D. Austin, A. Barbir, E. Peters, and S. Ross-Talbot, "Web services choreography requirements 1.0," World Wide Web Consortium, March 2004. [Online]. Available: http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311

[3] A. Arkin et al., "Web service choreography interface 1.0," 2003. [Online]. Available: http://www.w3.org/TR/wsci/

[4] N. Kavantzas et al., "Web Services Choreography Description Language 1.0," November 2005. [Online]. Available: http://www.w3.org/TR/ws-cdl-10/

[5] D. Martin et al., "OWL-S: Semantic Markup for Web Services," 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[6] E. Christensen et al., "Web Services Description Language (WSDL) 1.1," March 2001. [Online]. Available: http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[7] R. Chinnici et al., "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," Juni 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[8] M.-E. Cambronero, G. Daz, E. Martinez, and V. Valero, "A Comparative Study between WSCI, WS-CDL, and OWL-S." in *ICEBE*. IEEE Computer Society, 2009, pp. 377–382.

[9] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Let's Dance: A Language for Service Behavior Modeling," in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, vol. 4275. Springer, 2006, pp. 145–162.

[10] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

[11] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, "Business process execution language for web services version 1.1," 2007. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-bpel/

[12] F. Rosenberg, C. Enzi, A. Michlmayr, C. Platzer, and S. Dustdar, "Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL," in *EDOC*. IEEE Computer Society, 2007, pp. 15–26.

[13] pi4 Technologies Foundation, "Pi4soa," 2007. [Online]. Available: http://pi4soa.sourgeforge.net/

[14] "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5.

[15] W. M. van der Aalst and M. Song, "Mining social networks: Uncovering interaction patterns in business processes," *Business Process Management*, pp. 244–260, 2004.

[16] A. Barros, M. Dumas, and P. Oaks, "A Critical Overview of the Web Services Choreography Description Language (WS-CDL)," *BPTrends*, March 2005.