Automatic Extraction of Conversation Protocols from a Choreography Specification of Ubiquitous Web Services

Jonghun Park[1] and Byung-Hyun Ha[2]

[1] Dept. of Industrial Eng., Seoul National University, Seoul, 151-742, Korea
jonghun@snu.ac.kr
[2] Dept. of Industrial Eng., Pusan National University, Pusan, 609-735, Korea
bhha@pusan.ac.kr

Abstract. While web service technology is becoming a de facto standard for integration of business applications, it is also rapidly emerging as an effective means for achieving inter-operability among the devices in network centric ubiquitous systems. When such a web service enabled device engages in a conversation with a service provider, it becomes necessary to define an interaction logic required between them. For this purpose, one can use a choreography language to specify the rules of engagement between the device and the web service provider. This paper presents a framework for automatically synthesizing conversation protocols from a choreography description defined in WS-CDL. The proposed framework adopts WSCL as a conversation protocol language, and defines a set of rules that can be used to effectively transform a WS-CDL specification into WSCL documents for collaborating peers. It is expected that the work presented in this paper can enhance the interoperability between web service-based processes in ubiquitous systems through automating the process of extracting conversation protocols from a choreography definition.

1 Introduction

Web services are increasingly embedded in ubiquitous systems not only within a business network [1]. By embedding the web services into virtually any computing devices, it becomes possible for a device to discover and interoperate with other devices and remote services, establishing pervasive network of computers of all form factors and wireless devices. Currently, several ongoing efforts recognize the need for embedding web service capability into devices to enhance interoperability among them as well as with external services. These include Microsoft's invisible computing project [2], UPnP 2.0 [3], and OMA's OWSER [4]. Furthermore, recently proposed web service standards, such as WS-Discovery [5] and WS-Eventing [6], are also accelerating the wide deployment of web service technology into ubiquitous computing networks. In this paper, we collectively refer to the web services embedded in the devices of ubiquitous networks as ubiquitous web services.

X. Zhou et al. (Eds.): EUC Workshops 2006, LNCS 4097, pp. 122–132, 2006.
© IFIP International Federation for Information Processing 2006

Considering the importance of interoperability in realizing the ultimate vision of ubiquitous computing, we envision that making devices web service enabled appears to be a vital approach, and at the same time it necessitates several new breed of research problems to be addressed. In particular, when a device is web service enabled and engages in a conversation with a service provider, it becomes necessary to define an interaction logic required for them in order to coordinate the interactions during autonomous web services conversations. For this purpose, one can use a choreography language to specify the rules of engagement between the device and the web service provider.

There are currently two major approaches to describing web services choreographies, namely the *global view* approach and the *individual view* approach, depending upon whether they describe either (i) the choreography of an entire system consisting of all potential participants, or (ii) the choreographies expected by each individual participant [7]. WS-CDL (Web Services Choreography Description Language) [8] is a language that takes the global view approach whereas WSCL (Web Services Conversation Language) [9] is one that bases on the individual view approach. Indeed the individual view represents the expected conversation behavior of a single collaborating party, and it describes public aspects of a web service, leaving private aspects such as business logic to specific implementations. For this reason, we call it *conversation protocol* to distinguish it from the global choreography specification.

We identify two conversion problems associated with the approaches mentioned above: The first issue is how to obtain a global choreography specification from the conversation protocols of individual participants. The conversation protocols that specify the interactions among collaborating parties are usually developed independently with an expectation that they can function together, and the problem is to develop a method by which the independently defined conversation protocols can be composed together into a single global choreography. This approach requires the use of a global model that describes the desired flow of messages among the participants, and there have been several research results reported in the literature for this problem [10].

The second issue, which corresponds to the opposite case of the first issue, is to develop a framework that can generate a conversation protocol for each of the collaborating parties from a global choreography definition. The generated conversation protocol can then be used as an end point skeletal behavior description for building an executable process that will be used by a participant to seamlessly interact with the other collaborating processes. Yet, presently there is no concrete computational method available to address this problem.

Motivated by this, this paper considers WS-CDL as a global choreography specification language, and proposes a new framework that can support automatic synthesis of conversation protocols described in WSCL from a WS-CDL specification. Our proposed framework for automatic extraction of conversation protocols is expected to not only enhance the interoperability of the web service based interactions but also increase the user acceptance of recently emerged proposals for web service choreography in the emerging ubiquitous service networks.

The rest of the paper is organized as follows: Section 2 reviews the basic concepts of WS-CDL and WSCL, and then introduces an example scenario that motivates the presented research. In Section 3 we present the proposed framework, and it is demonstrated through the example scenario in Section 4. Finally, Section 5 gives the conclusion and descriptions on the future work.

2 Languages for Web Service Choreography and Conversation Specification

WS-CDL is an XML-based language that describes peer-to-peer collaborations of web service participants through defining, from a global viewpoint, their common and complementary observable behavior, where ordered message exchanges result in accomplishing a common goal [8]. It introduces various components to effectively describe observable behavior of multiple participants within the context of a global model. In order to define a choreography of interactions, one must first define data components such as role types, relationship types, information types, token types, and channel types, and declare necessary variables. Subsequently, a choreography definition is completed by specifying interactions and control flows among them in terms of ordering structures and work units.

On the other hand, WSCL allows the abstract interfaces of web services, i.e. the conversation protocols, to be defined. It specifies the XML documents being exchanged as well as the allowed sequencing of these document exchanges. The conversation proceeds from one interaction to another according to the legally defined transitions. The purpose of WSCL is to provide and define the minimal set of concepts necessary to specify conversations. Hence, WSCL describes the public processes in which the participants of a web service engage, and it focuses on the conversation behavior based on the individual participant's view.

Transformation of the data components of a WS-CDL document into the corresponding elements of WSCL for the participant in consideration can be carried out without difficulty if one uses an XML transformation language such as XSLT [11]. This transformation process can be considered as a projection into an individual view out of a global choreography. However, it is not straightforward to transform the control flows defined in a WS-CDL document into those available in WSCL. WS-CDL is rooted on pi-calculus and thus able to provide high-level control flow constructs such as parallel flows and loops whereas WSCL is based on automata formalism where only the finitely many states and transitions are allowed. Therefore, the proposed framework focuses on the problem of transforming the control flows of WS-CDL to those of WSCL, leaving out the other details such as the projection and transformation of XML elements.

As a motivating example, we consider the following scenario: A presentation room is equipped with an audio system and a video system, each of which provides a web service for setting up the corresponding equipment. When a user with a mobile computing device such as PDA and smart phone enters the room, the device needs to go through several interactions with the software agent

```
<choreography name="PresentationSetup" root="true">
  ...
<sequence>
  <interaction name="Login" ...> ...
    <exchange name="sendId"/>
  </interaction>
  <parallel>
    <interaction name="AudioSetup" ...> ...
      <exchange name="exchangeAudioInfo"/>
    </interaction>
    <workunit name="Repeat until video setup completes" repeat="vSetupDone = false">
      <choice>
        <interaction name="ProjectorSetup" ...> ...
          <exchange name="exchangeProjectorInfo"/>
        </interaction>
        <interaction name="CurvedWallSetup" ...> ...
          <exchange name="exchangeCurvedWallInfo"/>
        </interaction>
      </choice>
    </workunit>
  </parallel>
</sequence>
</choreography>
```

Fig. 1. A WS-CDL specification for the example scenario

responsible for coordinating the equipment in the room in order to configure the audio and video system appropriately.

More precisely, after the login interaction that certifies the mobile device, the setup processes for the audio and video systems are executed in parallel. We consider two available video systems, namely an LCD projector and a curved wall for setting up a virtual reality environment, from which the user can choose. As for the video system, the user is allowed to perform repeated trials for the equipment configuration. We further assume that the choreography requirement is defined by use of WS-CDL whereas the mobile device is configured to carry out conversation based on WSCL. Since WS-CDL itself is not an executable specification, the generation of a WSCL document is necessary for the mobile device and the software agent to interoperate with each other.

Based on the above choreography descriptions, a WS-CDL specification can be defined as shown in Figure 1. For the sake of brevity, only the XML elements pertaining to the control flows are shown in Figure 1. The most outer flow control construct is `sequence` that contains one `interaction` element and one `parallel` element which further includes an `interaction` and `workunit`. The behavior of the loop is controlled by use of the variable `vSetupDone` that is initially set to 'false' and then changed to 'true' when the result of `ProjectorSetup` or `CurvedWallSetup` is successful.

3 Proposed Framework

Having introduced the basic notions of WS-CDL and WSCL, we proceed to introduce the proposed approach to automatically synthesizing a WSCL specification from a given WS-CDL document. For the framework descriptions, we only consider the conversations between two parties out of many possible conversations

defined for multi-parties in WS-CDL, since WSCL can only support peer to peer interactions. Such a transformation for the parties in consideration can be easily carried out by use of projections into specific participants and roles as mentioned in Section 2.

We first define the notion of *conversation module* (CM) that represents a component to be composed. CM is a valid WSCL specification with the constraints that it has a single `Interaction` of type S and a single `Interaction` of type F where S and F respectively indicates the start and the finish of the module. The initial step of the proposed framework is to define the following WSCL elements for each `exchange` element of WS-CDL, E_i.

```
<Interaction StepType="" id="I_i"> ... </Interaction>
<Interaction StepType="" id="S_i"> ... </Interaction>
<Interaction StepType="" id="F_i"> ... </Interaction>

<Transition>
  <SourceInteraction href="#S_i"/> <DestinationInteraction href="#I_i"/> ...
</Transition>
<Transition>
  <SourceInteraction href="#I_i"/> <DestinationInteraction href="#F_i"/> ...
</Transition>
```

The above transformation introduces three WSCL `Interaction`s, I_i, S_i, F_i, which respectively represent the considered WS-CDL `exchange` element, and the start and finish of a WSCL `Interaction`. It also creates two WSCL `Transition`s among them. Furthermore, the WS-CDL's `interaction` element which the `exchange` belongs to is also recorded as an additional element so that it can be referred to when traversing the `Interaction` elements of a WSCL specification during the parallel composition defined later in this section. Note that the transformation of each WS-CDL `exchange` element yields a valid CM.

Once the transformation is carried out for all the `exchange` elements defined in the WS-CDL, the next step is to construct CMs for the `interaction`s defined in the WS-CDL, which may have more than one `exchange` element. Two CMs, CM_i and CM_j, that correspond to the `exchange` elements contained in an `interaction` of WS-CDL, are composed into another CM_k by using the following rule for sequential composition:

1) Remove the type F `Interaction` and its associated `Transition`s from CM_i
2) Remove the type S `Interaction` and its associated `Transition`s from CM_j
3) Introduce new `Transition`s from all the `Interaction`s that are the sources of the `Transition`s removed in Step 1) to the `Interaction`s that are the destinations of the `Transition`s removed in Step 2)

Subsequently, the following set of rules are defined in order to effectively transform the WS-CDL control constructs to the corresponding flow logic of WSCL. First, the `sequence` composition of two CMs, namely CM_i and CM_j, is computed by use of the above sequential composition rule. Second, the following rules are used to carry out `choice` composition of two CMs, CM_i and CM_j.

1) Remove all the `Interaction`s of type S and their associated `Transition`s from CM_i and CM_j

2) Remove all the `Interaction`s of type F and their associated `Transition`s from CM_i and CM_j
3) Introduce a new `Interaction` of type S, S_k, and new `Transition`s which connect from S_k to all `Interaction`s that are the destinations of the `Transition`s removed in Step 1)
4) Introduce a new `Interaction` of type F, F_k, and new `Transition`s which connect from all `Interaction`s that are the sources of the `Transition`s removed in Step 2) to F_k

Third, the CM of which the looping behavior is controlled by `workunit` is transformed into the corresponding WSCL elements by use of the following rules. We let T_s and T_f respectively be the set of `Transition`s of which the source is the `Interaction` of type S, and the set of `Transition`s of which the target is the `Interaction` of type F.

1) Introduce new `Transition`s that connect from all the `Interaction`s that are the sources of T_f to all the `Interaction`s that are the destinations of T_s
2) For each `Transition` introduced in Step 1), record the corresponding `exchange` element from the WS-CDL document from which `Interaction` is defined

Lastly, for the `parallel` composition, we construct a shuffle of two CMs, which generates all the possible interleavings between two finite state machines represented by CM_i and CM_j. That is, while the shuffle obeys the interaction order specified within each CM, it combines two interaction sequences in all possible combinations. During the `parallel` composition, each `Interaction` is augmented with an *execution history* defined as follows in order to identify the next executable `Interaction` as well as the destinations of its `Transition`s for repetition.

Definition 1 (Execution History). *An execution history is a set of* `Interaction`*s defined in CMs such that if two* `Interaction`*s in the execution history, I_k and I_l, belong to the same CM, then the following two properties hold:*

a) I_k and I_l can be connected by using only the `Transition`*s associated with the* `Interaction`*s in the execution history*
b) There is no `Interaction`*, I_m, in the execution history such that both of I_k and I_l are the destinations of outgoing* `Transition`*s of I_m*

From an execution history, we can construct a companion set, named *exchange history*, in view of WS-CDL specification, which is defined as follows.

Definition 2 (Exchange History). *Given an execution history containing the set of* `Interaction`*s that have been executed, the exchange history is defined as a set of* `exchange`*s (originally defined in WS-CDL) to which the* `Interaction`*s of the execution history correspond.*

Given an execution history, the set of next executable `Interaction`s is characterized by use of the notion of *reachability*, which is formally defined as follows.

Definition 3 (Reachability). *Let τ be the set of* `Transition`*s associated with an execution history such that i)* `Transition`*s in τ are not defined for modeling repetitions, and ii) they are not associated with the* `Interaction`*s of type F in the execution history. An* `Interaction`*, I_k, is said to be reachable from the execution history if the following three conditions are satisfied:*

a) I_k is not a member of the execution history
b) I_k is connected from one of the type S `Interaction`*s via the* `Transition`*s in τ*
c) The set defined by adding I_k to the given execution history is also an execution history

Based on the above definitions, the synthesis procedure for the `parallel` composition is described below in three phases: merging, repetition handling, and post-processing. The first phase that merges two CMs is formally defined as follows:

1) Introduce a new `Interaction` of type S, S_k, and annotate it with an empty execution history
2) For each I that is newly introduced from the previous step, perform Step 3) and go to Step 8)
3) For each `Interaction`, I_r, defined in the given CMs, perform Step 4) through Step 7), if I_r is reachable from the execution history of I
4) Introduce a new `Interaction`, I_n, for each I_r, and record the original `exchange` element corresponding to I_r in I_n
5) Annotate I_n with the execution history generated by adding I_r to I's execution history
6) If there exists an `Interaction`, I_e, of which the original `exchange` element and the exchange history are identical to those of I_n, then replace I_n with I_e
7) Add a new `Transition` from I to I_n
8) If there is an `Interaction` that is newly introduced, go to Step 2)

We remark that, in Step 4) of the above merge procedure, recording of the original `exchange` element defined in the WS-CDL document is necessary to relate an `Interaction` element of a WSCL specification with the `exchange` element it refers to. Next, the `Transition`s defined for handling the loops are taken care of by the following procedure.

1) For every `Transition` in the given CMs, T_r, that is defined for modeling a loop, carry out Step 2) with E_r, the set of `exchange` elements recorded in T_r
2) For each `Interaction`, I_s, whose original `exchange` is identical to that of T_r's source `Interaction`, carry out Step 3) and Step 4)
3) Add the `Transition`s from I_s to each `Interaction` I_t where I_t is an `Interaction` such that the result of subtracting I_t's original exchange from I_t's exchange history is identical to that of subtracting E_r from I_s's exchange history
4) Record E_r for all the `Transition`s added in Step 3)

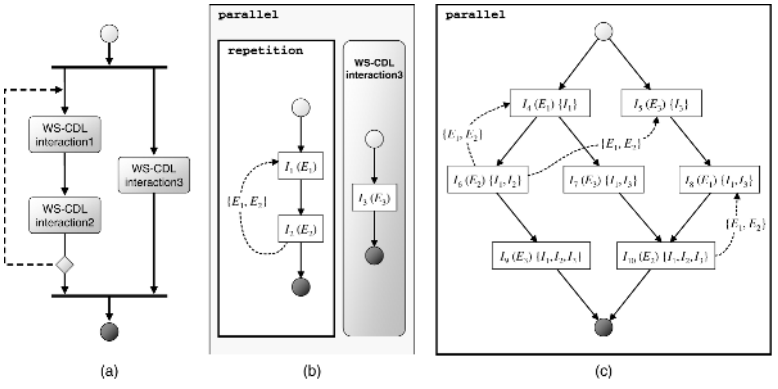

Fig. 2. An example illustrating the synthesis procedure when the parallel composition is performed for the `Interaction` element with a loop

Finally, we define the following post-processing procedure to produce a valid WSCL document.

1) Introduce a new `Interaction` of type F, F_k
2) Introduce `Transitions` from every `Interaction` which has no outgoing `Transition` to F_k
3) Remove the CMs that are given as an input and clear the annotations from all `Interactions`

The process of parallel composition becomes complex particularly when it is applied to the `Interaction` element that has a loop. In order to more clearly explain the procedures for the loop transformation in conjunction with parallel composition, we consider the example shown in Figure 2.

Given a WS-CDL specification shown in Figure 2 (a), we obtain the partial result shown in Figure 2 (b) after applying the transformation procedure defined for `workunit`. In Figures 2 (b) and (c), each CM is depicted as a state transition diagram with two circles that respectively represent the `Interactions` of types S and F and with rectangles that represent the other `Interactions`. A `Transition` is depicted as an arrow while it is shown as a dotted arrow if it is defined for modeling a loop. Furthermore, an `Interaction` other than those of the types S and F is also associated with a label that denotes the name of the `Interaction` as well as the name of the original `exchange` element it corresponds to. We name the `Interactions` as I_1, I_2, and I_3, and their original corresponding `exchanges` in the WS-CDL document as E_1, E_2, and E_3, respectively.

As shown in Figure 2 (b), the procedure for `workunit` transformation results in the `Transition` introduced to model the loop and the associated `exchange` elements recorded for it (i.e., $\{E_1, E_2\}$). We then apply the first phase of the parallel composition, and obtain the intermediate result that can be constructed by removing the dotted arrows and the `Interaction` of type F and its incoming transitions from Figure 2 (c). The execution histories computed during the first

phase are recorded as an annotation in the `Interaction`. For instance, I_7 in Figure 2 (c) has the execution history of $\{I_1, I_3\}$ associated with it.

As an example to discuss Steps 3) through Step 7) defined in the first phase of the parallel composition, consider I_{10} created from I_7 (with the execution history $\{I_1, I_3\}$) for the first time, due to the fact that I_2 of Figure 2 (b) is reachable from the execution history $\{I_1, I_3\}$. The original `exchange` element of I_{10} becomes E_2 since the I_2's original `exchange` is E_2 and the execution history of I_{10} is computed by adding I_2 to the execution history of I_7. Hence, the exchange history of I_{10} then becomes $\{E_1, E_2, E_3\}$. Continuing with I_8, we find that a newly introduced `Interaction` has the same set of original `exchange` elements and the exchange history as I_{10}. Consequently, this new interaction is removed, and the transition between I_8 and I_{10} is added instead. That is, this example signifies the fact that, when the exchange histories and the corresponding `exchange` elements are same for two `Interaction`s in WSCL, they can be modeled as a single `Interaction`.

Step 3) of the second phase of the parallel composition is essentially to invalidate the `exchange` elements that have already been executed, through identifying the `exchange` elements that can be re-executed when a repetition occurs. For instance, I_{10} is a state in which each of `exchange`s, E_1, E_2, and E_3 have been carried out at least once, and the `Transition` from I_{10} to I_8 represents that, the E_1 and E_2 are required to be performed again once the repetition occurs. Note also that, the execution of E_3 is prohibited after E_3 has been executed once. The resulting WSCL after applying all three phases of the parallel composition is shown in Figure 2 (c).

Given a WS-CDL document that is tree-structured, the proposed synthesis procedure starts from the leaf elements of the document, and then recursively applies the procedures mentioned above to each parent node in the document according to the type of the composition in consideration. We remark that the traversal order by which the procedures are applied does not change the final result.

4 An Example Application

In this section, we demonstrate the effectiveness of the proposed approach through an example. We apply the proposed synthesis procedure to the example WS-CDL shown in Figure 1, and obtain the WSCL specification with 10 `Interaction`s (two being the `Interaction`s of type S and F) and 17 `Transition`s after removing the interactions of type S and F from the resulting CM.

The detailed steps as well as the intermediate results for generating a WSCL specification out of a WS-CDL document are illustrated in Figure 3. First, Figure 3 (a) shows the given WS-CDL from which the initial CMs are derived as illustrated in Figure 3 (b). For the sake of simplicity, we rename the original `exchange` elements of the WS-CDL as follows: E_1 for the 'sendId' `exchange` element of 'Login' `Interaction` of Figure 1, and E_2 for the 'exchangeAudioInfo' `exchange` element of 'Audio Setup' `Interaction` of Figure 1, and so on.

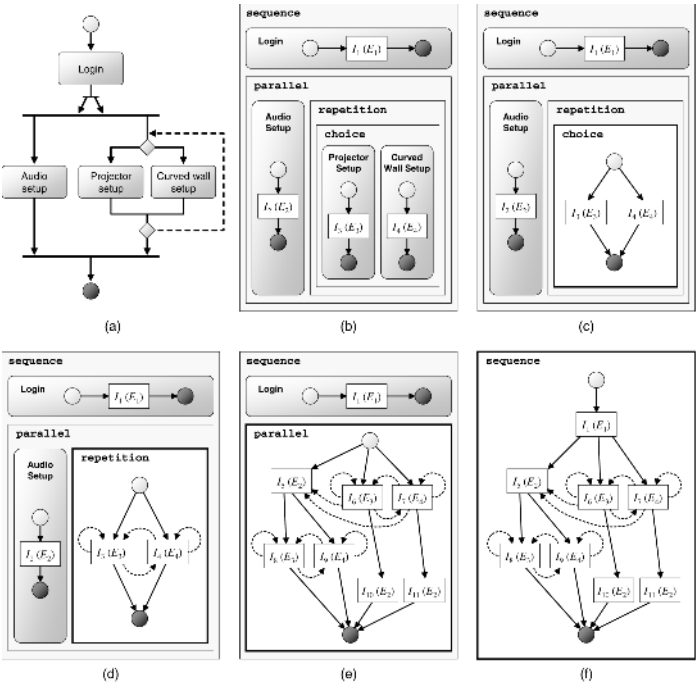

Fig. 3. Illustration of steps for automatically generating a WSCL specification

For each CM, the `Interaction`s of type S and F are represented as circles while the other `Interaction`s are represented as rectangles with a label denoting the name of the `Interaction` and the name of the original `exchange` element it corresponds to. For instance, $I_9(E_4)$ of the step (e) indicates that the name of `Interaction` is I_9 and the original `exchange` element in WS-CDL it refers to is E_4 (i.e., 'exchangeCurvedWallInfo' element in Figure 1). Furthermore, `Transition` elements of WSCL are represented as arrows except those `Transition`s introduced to model the loop, which are represented as dotted arrows.

Figure 3 (c) shows the intermediate result after applying the rules defined for the choice composition, and Figure 3 (d) shows the result of transforming the `workunit` control structure. Finally, Figures 3 (e) and (f) respectively show the results obtained after the parallel and sequence compositions are performed.

5 Conclusion

The recently emerged web services choreography languages such as WS-CDL and WSCL represent significant steps towards building platform-independent, distributed, flexible web service applications. In this paper, we addressed the problem of automatically generating a conversation protocol that represents the

individual participant's view on a given global choreography definition. The conversation protocol specification obtained by the proposed framework describes the public behavior of the considered endpoint, and therefore can serve as a skeleton for building executable process, supporting seamless interoperability between the participants of interacting web services in ubiquitous networks. Future work will aim at implementing the proposed framework by use of XSLT patterns that effectively model the synthesis rules presented in the paper. Another future work will be to apply the proposed approach to other classes of conversation protocol languages.

Acknowledgments. This work was supported by the Korea Research Foundation Grant funded by the Korean Government(MOEHRD) (KRF-2005-041-D00917).

References

1. Sashima, A., Izumi, N., Kurumatani, K.: Location-mediated coordination of web services in ubiquitous computing. In: Proceedings of the IEEE Intl Conf. Web Services (ICWS04). (2004)
2. Microsoft: The microsoft invisible computing project. Web site, Microsoft (2006) http://research.microsoft.com/invisible/.
3. UPnP: The UPnP forum. Web site, UPnP (2006) http://www.upnp.org.
4. OMA: OMA web services enabler (owser): Overview. Document, OMA (2004) http://www.openmobilealliance.org/.
5. Beatty, J., et al.: Web services dynamic discovery (ws-discovery). Specification, Microsoft (2005) http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-Discovery.pdf.
6. Bank, D., et al.: Web services eventing (ws-eventing). Specification, BEA (2004) http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf.
7. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. In: Proceedings of the First International Workshop on Web Services and Formal Methods. (2004)
8. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language Version 1.0. W3C candidate recommendation, World Wide Web Consortium (2005) http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.
9. Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., Williams, S.: Web Services Conversation Language (WSCL) 1.0. W3C note, World Wide Web Consortium (2002) http://www.w3.org/TR/2002/NOTE-wscl10-20020314/.
10. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Computing **8**(6) (2004) 51–59
11. Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C recommendation, World Wide Web Consortium (1999) http://www.w3.org/TR/xslt.