A comparative study between WSCI, WS-CDL, and OWL-S *

María Emilia Cambroner, Gregorio Díaz, Enrique Martínez, and Valentín Valero
Escuela Politécnica Superior de Albacete. Universidad de Castilla-La Mancha
Departamento de Sistemas Informáticos
Campus Universitario s/n. 02071. Albacete, SPAIN
{MEemilia.Cambroner,Gregorio.Diaz,Valentin.Valero}@uclm.es
{emartinez}@dsi.uclm.es

Abstract

Choreography languages allow us to describe Web Services compositions from a global viewpoint in Service Oriented Architectures (SOA). However, none of the existing languages has achieved the status of de facto standard for that purpose until now. In this paper we compare three existing proposals to specify Web Services choreographies: WSCI, WS-CDL, and OWL-S. First, we describe the main characteristics of each one of these languages, and after that we compare the different structures of the three languages. Finally, we present some conclusions of our work.

1. Introduction

The importance of Service-Oriented Architectures (SOA) has grown in the last years because they allow the integration of software applications between different organizations. In these architectures, applications are exposed as services, and these services are interconnected through the use of a set of standards (SOAP, UDDI, WS-Security, ...). This is the reason because standardization is one of the main aspects of SOA. While a certain level of maturity has been achieved in the adoption of standards to interconnect and describe Web Services, there are still challenges related to the business processes executed by Web Services compositions.

The terms orchestration and choreography refer to two ways of describing Web Services compositions. Orchestration languages always represent the composition from the viewpoint of the parties involved in this composition. WS-BPEL is the most adopted language for that purpose.

On the other hand, the target of choreography languages is the coordination of long-running interactions between multiple distributed parties, where each one of the parties uses Web Services to offer his externally accessible operations. Choreography languages depict the composition from a global viewpoint, showing the interchange of messages between parties. However, there is not an only standard that has been widely adopted for that purpose until now.

Our goal with this paper, then, is to present a comparative study of three existing languages to specify Web Services choreographies: Web Service Choreography Interface (WSCI, [1]), Web Service Choreography Description Language (WS-CDL, [2]), and Ontology Web Language for Services (OWL-S, [3]).

The rest of the paper is structured as follows: Section 2 shows a general description of WSCI language. Section 3 explains the main features of WS-CDL language. Section 4 provides a brief description of the DAML program and the OWL-S language. Section 5 is devoted to the comparison of the different structures of these languages. Finally, in Section 6, some conclusions are presented.

2. Web Service Choreography Interface (WSCI)

The Web Service Choreography Interface (WSCI, [1]) is an XML-based language to describe the interface of a Web Service participating in a choreographed interaction with other services. This interface shows the flow of messages exchanged by the Web Service. The language has been developed by companies like Sun, SAP, BEA and Intalio.

A WSCI interface describes the observable behavior of only one Web Service. This behavior is expressed by means of temporal and logical dependencies in the flow of messages. For that purpose WSCI includes sequencing rules, correlation, exception handling, and transactions. WSCI also describes the collective message exchange among the
Web Services participating in the choreography, providing a global view of the interactions. Therefore, a WSCI choreography consists of a set of interfaces, one for each Web Service taking part of it, as we can see in Figure 1.

**Figure 1. WSCI architecture**

*Contexts* are used to describe the environment within which a set of activities is executed. These contexts include the set of declarations available to the activities, the set of possible exceptions and the behavior related to these exceptions, and the transactional properties of the activities, including the compensations to undo these activities.

WSCI uses a mechanism called *correlation* to associate a message with a concrete conversation. Multiple conversations can be distinguished through the use of different correlation instances. Properties of a concrete correlation are communicated as part of messages exchanges.

### 3. Web Service Choreography Description Language (WS-CDL)

The Web Service Choreography Description Language (WS-CDL, [2]) is an XML-based language to describe peer-to-peer collaborations of Web Services taking part in a choreography. This description defines, from a global viewpoint, the common behavior of the services, and the ordered message interchanges to make reaching a common business goal possible.

The goal of specifying Web Services choreographies is composing peer-to-peer interactions between any kind of services, regardless of the programming language or the environment that host the service. In WS-CDL the collaboration between Web Services takes place within a set of agreements about the ordering and constraint rules.

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

### 4. Ontology Web Language for Services (OWL-S)

The Ontology Web Language for Services (OWL-S, [3]) was originally known as DAML-S. The objective of the DARPA Agent markup Language (DAML) program is the development of a language and tools that facilitate the concept of Semantic Web. As part of this program, the Web Services ontology OWL-S has been developed. The aim of this ontology is to automate the discovery, invocation, composition, interoperation and monitoring of Web Services. This ontology has been developed by Carnegie Mellon University, Nokia, Stanford University, . . .

In Figure 2 we can see the ontology for Web Services proposed by OWL-S. This ontology is based on providing three essential kinds of information about the services:

**Figure 2. Web Services ontology**

- *What does the service provide?* This information is given by the **Service Profile**.
- *How is the service used?* This information is given by the **Service Process Model**.
- *How to access the service?* This information is provided by the **Service Grounding**.

Briefly, the Service Profile provides the information that agents need to discover the service, while the Service Process Model and the Service Grounding give the information that agents need to use the service.

Although OWL-S defines an ontology for each one of these three areas, it also allows the definition of alternative approaches. OWL-S also defines another ontology for the required resources. This ontology covers the description of physical resources, temporal resources and computational resources related to the services.
5 Comparison

5.1 Basic Structures

WSCl In WSCI the basic activities are called atomic activities and the action element is the main one. This element describes the way in which Web Services use an elementary operation within a context, e.g., the exchange of a message with another Web Service. The operation attribute can be used to reference a Web Services Description Language (WSDL) operation that the action performs.

The role attribute is an optional attribute that associates an action with a role name. It can be used to reference the definition of a role given by some other specification.

The correlate element is used to relate an action to a correlation definition. It serves to indicate in which particular execution context is performed the action, allowing us to correlate a message with a particular conversation. The correlation attribute is mandatory and it references a particular correlation specification.

The call element is used to indicate the activities that will happen while an action that handles a request-response operation is performed by a Web Service. This element is forbidden for all WSDL operations apart from request-response (the service receives a message and sends a response).

WS-CDL In WS-CDL the basic building block of a choreography is the interaction element. It indicates information exchanges between participants, possibly including the synchronization of some information values. These interactions are performed when one participant sends a message to another participant in the choreography. When the message exchanges complete successfully, the interaction completes normally. The channelVariable attribute specifies the channel variable used to do the communication during the interaction and the operation attribute specifies the name of operation that is associated with the interaction.

The participant element specifies the relationship type the interaction participates in, and the requesting and accepting participants.

The exchange element is used to exchange information during the interaction. The action attribute specifies the direction of the exchanged information, i.e., request or respond.

The send element and the receive element inside the exchange element indicate that information is sent from a participant or information is received at a participant respectively. These elements can also specify the variables exchanged, and if an exception must be thrown.

The timeout element allows us to specify the maximum amount of time to complete an interaction, by means of the time-to-complete attribute. When this time is exceeded, a timeout occurs. This element also allows us to modify some records in both participants when the timeout occurs.

Finally, the record element is used to create or change the value of one or more variables.

OWL-S In OWL-S services are modeled as processes. These processes are specifications of the ways clients may interact with services. For that purpose, OWL-S includes a subclass of the Service Model called Process.

We can distinguish three different kinds of processes: atomic processes, simple processes, and composite processes. Composite processes correspond to activities that require multiple service interactions, so we only talk about atomic and simple processes in this section.

Atomic processes are executed in a single step and never have subprocesses. They just receive an input message, do some work, and finally send an output message. There are always only two participants for that kind of process, the client and the server.

Simple processes have also a single step execution. They are used as abstractions, providing a view of some atomic process. In this case, the simple process is realizedBy the atomic process.

Finally, we must take into account that processes can have two different goals: They can return some new information based on some given information (using inputs and outputs) or they can produce a change in their environment (using preconditions and effects). There are several classes defined in the OWL-S model related to these four elements.

Discussion The three languages have basic structures to describe the message exchange between parties in a composition, but there are several differences between the elements used for that purpose. While the interaction element in WS-CDL allows us to exchange multiple messages between two parties (in both directions), the action element in WSCI and the atomic process in OWL-S refer to a single exchange.

The interaction element in WS-CDL pays special attention to the variables exchange between the different parties in each exchange element, while the action element in WSCI only specifies the operation performed by the message. In OWL-S there is a list of inputs and outputs related to each atomic process.

Finally, each action element in WSCI only specifies one of the roles participating in the exchange, the sender or the receiver. The connect element is used in the global model to relate a send message to a receive message from different interfaces. The interaction element in WS-CDL specifies both roles, indicating which one is the requesting participant and which one is the accepting participant. In OWL-S the atomic processes always have two properties to indicate which role is the client of the service and which role is the server. In this aspect, WS-CDL and OWL-S are more powerful than WSCI, in the sense that they need less code to express a collaboration between two parties.
5.2 Complex Structures

**WS-CI** In WSCI, *complex activities* contain a set of activities and define the order in which these activities are performed. A complex activity can contain one or multiple activity sets. Next, we are going to see in more detail each one of the complex activities that can be used:

- The **all** activity performs the whole set of activities that contains in any order, possibly in parallel.
- The **sequence** activity performs all the activity set in sequential order.
- The **choice** activity performs only one activity set from the collection of multiple activity sets within this complex activity. The decision is made based on events. The event can be the reception of a message, the expiration of a timeout, or the throwing of a fault. When multiple events overlap, there is no way to know which one of the possible activity sets is executed.
- The **foreach** activity executes the activity sets within repeatedly. The **select** attribute is an expression that evaluates to a list of items. The activity set is repeated once for each item in this list. If the list is empty, the activity set is not performed.
- The **switch** activity selects one activity set from the collection of multiple activity sets within this complex activity based on the evaluation of conditions. All the **case** elements inside are mutually exclusive, selecting the corresponding activity set if the value of the condition is true for that case. Only one case can be executed, so if multiple case elements can be performed, the first one in the definition has the biggest priority. If no other condition is fulfilled, the activity set within the **default** element is performed.
- The **until** activity performs the activity set that contains repeatedly based on a Boolean condition. The until activity is repeated one or more times because the condition is evaluated after each iteration of the activity set. If false the activity set is repeated, otherwise the activity ends.
- The **while** activity performs the activity set that contains repeatedly based on a Boolean condition. The while activity is repeated zero or more times because the condition is evaluated before each iteration of the activity set. If true the activity set is executed, otherwise the activity ends.

**WS-CDL** In WS-CDL, we can distinguish two different kinds of *complex activities* inside a choreography: the workunit element and the ordering structures.

The **workunit** element specifies a condition that must be fulfilled in order to perform some work and/or the repetition of some work. It completes successfully when the set of activities inside completes successfully. The optional attribute **guard** specifies the condition that must be fulfilled to perform the workunit, whereas the optional attribute **block**, with false value as default, indicates whether the element have to block waiting for the “true” evaluation of the guard condition or it skips the activities inside when the guard condition evaluates to “false”. The optional attribute **repeat** specifies the repetition condition of the workunit and it is always not blocking. When there is not guard condition specified, then it is considered to be always true, while when there is not repetition condition specified, then the workunit is not considered to be executed again after one execution.

The **ordering structures** are used to combine basic activities and other complex activities in a nested way, expressing the order in which actions are performed within the choreography. There are three ordering structures:

- The **sequence** ordering structure expresses that the set of activities inside must be executed sequentially.
- The **parallel** ordering structure indicates that the set of activities inside must be executed concurrently. It completes successfully when all the concurrent activities complete successfully.
- The **choice** ordering structure specifies that only one of multiple activities can be executed. If the choice have workunits inside, only the first one in lexical order with a “true” guard condition is selected. If there are other activities, there is no way to know which one is selected; it is considered as a non-observable decision.

**OWL-S** Composite processes in OWL-S contain other processes of any kind in a nested way. They also specify the way in which their contents are executed, such as sequence or any order. The **composedOf** property is used to specify the control construct corresponding to the composite process, i.e., the way in which their contents are executed. The control constructs provided by OWL-S are the following:

- The **Sequence** control construct specifies a list of subprocesses to be executed in a row. The **ControlConstructList** inside contains the list of subprocesses to be executed in sequence.
- The **Split** control construct specifies a set of subprocesses to be executed concurrently. This process completes as soon as all its subprocesses have begun their execution. The **ControlConstructBag** inside contains the set of subprocesses to be executed in parallel.
- The **Split+Join** control construct also specifies a set of subprocesses to be executed concurrently, but in this...
case the process completes when all its subprocesses have finished. Again, the ControlConstructBag contains the set of subprocesses to be executed in parallel.

- The Any-Order control construct specifies a set of subprocesses to be executed in any order but not concurrently. The process completes when all its subprocesses have finished. This control construct also uses the ControlConstructBag to specify the subprocesses that contains.

- The Choice control construct specifies a set of subprocesses and only one of them is executed. The selection criteria are non-observable, so any of the subprocesses can be chosen. This kind of composite process also uses the ControlConstructBag to specify all the possible subprocesses.

- The If-Then-Else control construct executes different subprocesses depending on the value of a condition. The ifCondition property specifies the condition we have to test. The then property specifies the subprocess we execute if the condition is “true”, whereas the else property specifies the subprocess we execute when the condition is “false”.

- The Repeat-While control construct iterates the execution of a subprocess while a condition evaluates to true. This condition is always evaluated before the execution. The whileCondition property specifies the condition we test, and the whileProcess property specifies the subprocess we execute repeatedly.

- The Repeat-Until control construct iterates the execution of a subprocess as long as a condition evaluates to true. This condition is always evaluated after the execution, so at least one execution is done. The untilCondition property specifies the condition we test after each execution, and the untilProcess property specifies the subprocess we execute repeatedly.

Discussion In Table 1 we show the equivalences between the different complex structures we have described corresponding to each language.

The sequence construction exists in the three languages with the same meaning, the sequential execution of the activities or processes within the construction.

The choice construction also exists in the three languages but there are some differences. While in WSCI the selection is based on the triggering of an event, in WS-CDL and OWL-S the selection criteria for the activities inside are non-observable. WS-CDL also allows us to use workunits within the choice, restricting the possible selections to the workunits that match their guard condition.

A construction to indicate the concurrent execution of several activities also exists in the three languages, but with different names. The all construction in WSCI indicates that the activities are executed in parallel or in any order (but not concurrently). The parallel construction in WS-CDL indicates that the activities are executed concurrently (the specification says nothing about a possible execution in any order but not concurrently). Last, in OWL-S we have three different constructions for that purpose: The Split process indicates the concurrent execution of all its subprocesses without waiting for the completion of these subprocesses, the Split+Join process also indicates the concurrent execution of all its subprocesses but it waits for the completion of them, and the Any-Order process indicates the execution in an undefined order of the subprocesses but not concurrently.

The switch construction, that selects one activity from a collection, only exists in WSCI, but it can be emulated in WS-CDL and OWL-S by using some other construction. In WS-CDL we can use multiple non-blocking workunits with the guard condition specified, while in OWL-S we have the If-Then-Else process and with several of these processes nested we can achieve the same behavior.

Both, the until construction in WSCI and the Repeat-Until process in OWL-S, indicate the repetition of the contents based on the truth value of a condition, where the condition is evaluated at the end of each iteration. This behavior can be emulated in WS-CDL by using a workunit with a repeat condition specified.

We also have the while construction in WSCI and the Repeat-While process in OWL-S indicating the repetition of the contents based on the truth value of a condition, but with the condition evaluated before each iteration. In this case, an emulation in WS-CDL can be done by using a workunit with a guard and a repeat condition specified (both evaluating the same condition).

Finally, the foreach construction only exists in WSCI and there is not any equivalent construction in the other two languages.

6. Final Discussion

Both, WSCI and WS-CDL, are W3C proposals, but WSCI last update was released in 2002, so it has not got received any attention in the last years. On the other hand, WS-CDL is the ongoing standardization initiative for Web Service choreography, but it has not achieved the status of being accepted as the de facto standard for that purpose. Apart from these two proposals, we have the OWL-S language as a part of the emerging Semantic Web, so its success is closely related to the consolidation of this framework worldwide in the future, which is not clear now.

As we can see in [4], some people think that the reason because none of the choreography standardization efforts has been adopted by a wide user base is that Service-Oriented Architectures (SOA) has not gained enough maturity until now. They think that some issues have to be solved
before we reach the adoption of a SOA infrastructure that integrates choreography (the identification of patterns for service interactions, the definition of a service interaction meta-model, . . .). However, the current choreography languages can be seen as a starting point to reach these goals. E.g., the elements of a service interaction meta-model will be very similar to the elements we have in WS-CDL.

Timing restrictions are used very often in the composition of Web Services, being a critical issue in real-time systems. For example, we want to indicate the amount of time we wait for the confirmation of a purchase order. In WS-CDL and WSCI time constraints can be specified by using the timeout element and the timeout event, respectively, but the specification of OWL-S says nothing about these restrictions. Nevertheless, several efforts have been devoted to extend OWL-S with a time ontology [5, 6].

The use of a formal language to describe a Web Service choreography facilitates the validation of compositions by applying validation techniques already defined for this formalism. Only WS-CDL of the three languages we are comparing is based on a formal language (π-calculus) [7], but there is not a clear translation from all the elements of WS-CDL into π-calculus, so we cannot apply any validation technique directly. The scientific community has developed multiple translations of these three languages into different formal representations [8, 9, 10]. However, all these proposals only take into account a subset of the elements of each language, so they cannot guarantee full correctness of the given specifications.

Concerning the relation with other standards, these three choreography languages are XML-based and can work together with the WSDL language, using this well-established standard to describe the Web Services participating in the composition. WS-CDL and WSCI do not cover the description and execution of the workflow corresponding to each service in the composition, so we are free to use different mechanism for each one of these services, such as WSFL and WS-BPEL. On the other hand, OWL-S intends to cover this work and extensions like OWL-WS (OWL for Workflow and Services, [11]) has been developed for that purpose. Finally, we also have to take into account that OWL-S builds on OWL (Ontology Web Language), so it makes use of some of the ontologies defined by this language.

### Table 1. Equivalences between complex structures

<table>
<thead>
<tr>
<th>WSCI</th>
<th>WS-CDL</th>
<th>OWL-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>sequence</td>
<td>Sequence</td>
</tr>
<tr>
<td>choice (events)</td>
<td>choice (non-observable)</td>
<td>Choice (non-observable)</td>
</tr>
<tr>
<td>all (concurrent or unspecified order)</td>
<td>parallel (concurrent)</td>
<td>Split, Split+Join, and Any-Order</td>
</tr>
<tr>
<td>switch</td>
<td>Multiple workunits with guard conditions</td>
<td>If-Then-Else</td>
</tr>
<tr>
<td>Until</td>
<td>workunit with guard and repeat conditions</td>
<td>Repeat-While</td>
</tr>
<tr>
<td>While</td>
<td>workunit with guard and repeat condition</td>
<td>Repeat-Until</td>
</tr>
<tr>
<td>foreach</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### References

1. A. Arkin et al. *Web Service Choreography Interface (WSCI) 1.0*. http://www.w3.org/TR/wsci/.


