Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 229 (2009) 155–174

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Formal Modeling and Conformance Validation for WS-CDL using Reo and CASM

Samira Tasharofi[a,b,1] Marjan Sirjani[a,b,2]

[a] *Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran*

[b] *School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM), Niavaran Square, Tehran, Iran*

Abstract

WS-CDL is a choreography language that describes peer-to-peer collaborations of participants by defining their common and complementary observable behaviors from a global viewpoint. 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 easy to automatically validate conformance and ensure interoperability. To this purpose, WS-CDL must be based on or related to a formal language that provides these validation capabilities. In this paper, we benefit from Reo and Constraint Automata with State Memory (CASM) to address this problem by providing a unified formalism for choreography and orchestration. Furthermore, we show how to exploit this formalism for conformance validation by giving proper definitions for end-point projection and conformance problem.

Keywords: Web services, Choreography, Orchestration, WS-CDL, Conformance, Reo, Constraint Automata with State Memory

1 Introduction

Service-Oriented Computing (SOC) is an emerging paradigm for distributed computing and e-business processing. The aim of web service composition is to provide the mechanism to fulfill the complexity of the execution of business processes. Two different but overlapping viewpoints for the composition of web services are under investigation, namely orchestration and choreography. The former focuses on a single service, describing its interactions with other services as well as its internal actions. The latter is a multi-party contract that describes the external observable behavior across multiple participants from a global view [24,25]. Following the second approach, the overall activity is achieved by the composition of peer-to-peer

[1] Email: stasharofi@ut.ac.ir

[2] Email: msirjani@ut.ac.ir

doi:10.1016/j.entcs.2009.06.034

interactions among the collaborating services. Therefore, choreography is a specification protocol defining a global picture of the way services interact with each other.

While several proposals exist for orchestration languages (e.g. BPML [1] and BPEL [22]), choreography languages are still in a preliminary stage of definition. The first proposed language, WS-CDL [3], was issued by the World Wide Web Consortium (W3C). 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 easy to promote a common understanding between participants (web services), automatically validate conformance, ensure interoperability, and increase robustness [2]. WS-CDL is neither an executable business process description language nor an implementation language [3]. As mentioned in [10], one of the goals of WS-CDL is using it in conjunction with BPEL for building service oriented systems in a complementary fashion. In this way, the conformance problem, which examines whether the behavior of an orchestration conforms to the protocol specification, is an important issue to be resolved for achieving this goal.

In this paper, we use Reo and Constraint Automata with State Memory (CASM) for modeling choreographies and conformance validation. Reo [5] is an exogenous coordination model wherein complex coordinators, called connectors, are compositionally built out of simpler ones. Constraint automata [8] and its variations, e.g., CASM [6], are proposed as operational semantics for Reo. The advantage of Reo is that Reo circuits can be used for modeling both communication and coordination of web services [18,27,28]. In [28] we proposed a mapping from BPEL to Reo and CA (simple form of CASM). Regarding to this merit of Reo and our previous work [28], in this paper our goal is to build up a unified formalism for both choreography and orchestration, and use it for automatic analysis of conformance in choreography and orchestration. We present a slightly modified version of CASM, CASM^{ch} customized to accommodate WS-CDL concepts for modeling choreographies. By translating each WS-CDL component to Reo and its corresponding CASM^{ch}, the operational semantics of the choreography is obtained in terms of CASM^{ch}compositionally. Then, we define end-point projection on CASM^{ch} which results in the behavior of a given party in a choreography in the form of CASM. Then we present the simulation relation in CASM and use it for conformance analysis of choreography and orchestration.

The structure of this paper is as follows: Section 2 contains a brief description of Reo and CASM. An overview of WS-CDL and our proposed model for WS-CDL components are provided in Section 3. The theory of addressing the end-point projection and the conformance problems are presented in Section 4, while its practical example appears in Section 5. Finally, the previous related works and our conclusions can be found in Sections 6 and 7, respectively.

2 Overview of Reo and Constraint Automata with State Memory

Reo is an exogenous coordination language based on a calculus of channels [5]. Components in Reo are connected via connectors which coordinate their activities. Primitive connectors are channels which have two ends. There are two types of channel ends: *source* and *sink*. A source channel end accepts data into its channel, and a sink channel end dispenses data out of its channel. The channel can be defined by users which allows an open-ended set of different channel types, each with its own policy for synchronization, buffering, ordering, computation, data retention/loss, etc. However, some basic types of channels, used in this paper are: Synchronous Channel (*Sync*), Synchronous Drain (*SyncDrain*), and Lossy Synchronous Channel (*LossySync*). Complex connectors are constructed through composition of simpler ones by applying *join* operation. Whereas, the internal behavior of complex connectors can be abstracted away by applying the *hide* operation. A component can write a data items to a *source* node that it is connected to. The write operation succeeds only if all source channel ends coincident on the node accept the data item, in which case the data item is written to every source end coincident on the node. A *source* node, thus, acts as a *replicator*. A component can obtain data items, by an input operation, from a *sink* node that it is connected to. A take operation succeeds only if at least one of the sink channel ends coincident on the node offers a data item; if more than one coincident channel end offers data items, one is selected nondeterministically. A *mixed* node nondeterministically selects and takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. A sink or mixed node, thus, acts as a nondeterministic *merger*.

Constraint automata (CA) [8] are proposed as compositional semantics for Reo, based on timed data streams [7]. Each element of a timed data stream is a pair of time and a data item, where the time indicates when the data item is being input or output. A transition fires if it observes data item in a port of the component and according to the observed data, the automaton may change its state. Therefore, the automata-states stand for the possible configurations, while the automata-transitions represent the possible data flow and its effect on these configurations.

Definition 2.1 (*Constraint Automata*). A constraint automaton is a tuple $A = (Q, \mathcal{N}ames, \longrightarrow, Q_0)$ where [8]:
Q is a finite set of states, $\mathcal{N}ames$ is a finite set of names (e.g. I/O ports of a component), $\longrightarrow$ is a finite subset of $Q \times 2^{\mathcal{N}ames} \times DC \times Q$, called the transition relation of A, and $Q_0 \subseteq Q$ is the set of initial states. DC is data constraint that plays the role of guard for transition. For example, $d\_A = d\_B$ is a data constraint that imposes the observed data on ports A and B must be equal.

The *join* operator in Reo, is implemented by the *product* operation in CA and defined as following:

Definition 2.2 (*Product Automaton*). The product-automaton of the two con-

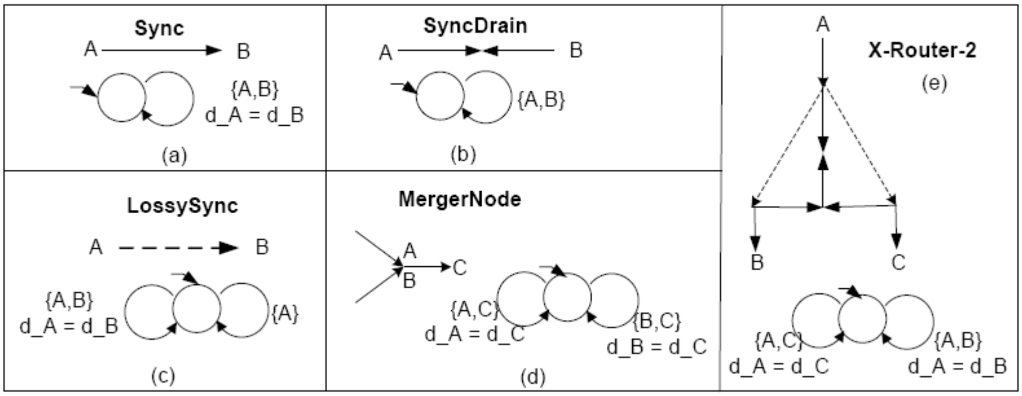

Fig. 1. Some basic Reo channels, merger node, X-Router-2 Reo connector, and their corresponding CA

straint automata $\mathcal{A}_1 = (Q_1, \mathcal{N}ames_1, \longrightarrow_1, Q_{0,1})$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}ames_2, \longrightarrow_2, Q_{0,2})$, is [8]:
$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N}ames_1 \cup \mathcal{N}ames_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$ where $\longrightarrow$ is defined by the following rules:

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 p_1,\quad q_2 \xrightarrow{N_2, g_2}_2 p_2,\quad N_1 \cap \mathcal{N}ames_2 = N_2 \cap \mathcal{N}ames_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle} \text{ and } \frac{q_1 \xrightarrow{N, g}_1 p_1,\quad N \cap \mathcal{N}ames_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle p_1, q_2 \rangle}$$

and latter's symmetric rule. □

Fig. 1 shows the the Reo channels which we used in this paper, a merger node, and a Reo connector which selects one of its two active outputs non-deterministically and thus, is called *X-Router-2* as well as their corresponding constraint automata. Another operation in CA, the *hide* operation [8], which corresponds to the *hide* operation in Reo, enables us to abstract away from internal nodes of a Reo circuit.

Ordinary CA are suitable for modeling only data-abstract coordination mechanisms. In many applications, the data-abstract view is too coarse and we need to reason about the values hold in each component in its different states. Constraint Automata with State Memory (CASM) are variant form of CA which enable us to model different states of each component along with the value of the states. The CASM used in this paper are a slightly simplified form of the CASM presented in [6].

Definition 2.3 (*Constraint Automata with State Memory (CASM)*). A constraint automaton with state memory is a tuple $\mathcal{A} = (Q, \mathcal{N}ames, \longrightarrow, Q_0, \mathcal{M}, \mathcal{V}_0)$ where:

- Q, $\mathcal{N}ames$, $\longrightarrow$, and Q_0 are defined the same as ordinary CA.
- $\mathcal{M}$ is a set of memory cells partitioned into (not necessarily disjoint) memory cells $\mathcal{M}_q$, for all $q \in Q$. For $m \in \mathcal{M}$, $q.m$ is used to uniquely identify the memory cells of the state $q \in Q$.
- For every $q \in Q$ the value function $\mathcal{V}_q : M_q \to Data$ is defined when the automaton is in state q. The set $\mathcal{V}_0$ consists of the initial value function $\mathcal{V}_{q_0}$, each of which gives the initial values of its corresponding initial state $q_0 \in Q_0$.

- The data constraint language is extended to include relativized memory cell names $s.m$ and $t.m$, for $m \in \mathcal{M}$ as well as the usual d_n for node names $n \in \mathcal{N}ames$ such that for each transition $q \xrightarrow{N,g} p$ the free variables of g are in the set $\{s.m | m \in \mathcal{M}_q\} \cup \{t.m | m \in \mathcal{M}_p\} \cup \{d\_n | n \in \mathcal{N}ames\}$. The special symbols s and t refer to the source and target states of a transition respectively.
- Given the data assignment $\delta_N : N \rightarrow Data$ and the value function $\mathcal{V}_q : \mathcal{M}_q \rightarrow Data$, a transition $q \xrightarrow{N,g} p$ is possible if there exists a value function $\mathcal{V}_p : \mathcal{M}_p \rightarrow Data$ such that $g[q/s, p/t]$ is true under the mappings δ_N, $\mathcal{V}_p$, and $\mathcal{V}_q$. Traversing the transition, makes $\mathcal{V}_p$ the value function for memory cells $\mathcal{M}_p$ of state p.

The product of two CASM that does not share common memory cells is defined similar to the product of two ordinary CA with two additional points as:

- For each state $\langle q_1, q_2 \rangle$ resulted from the product, $\mathcal{M}_{\langle q_1, q_2 \rangle} = M_{q_1} \cup M_{q_2}$ for all $q_1 \in Q_1$ and $q_2 \in Q_2$.
- The set $\mathcal{V}_0$ consists of the value function $\mathcal{V}_{\langle q_1, q_2 \rangle} : \mathcal{M}_{\langle q_1, q_2 \rangle} \rightarrow Data$, for $q_1 \in Q_{0,1}$ and $q_2 \in Q_{0,2}$, defined using $\mathcal{V}_{q_1} \in \mathcal{V}_{q_{0,1}}$ and $\mathcal{V}_{q_2} \in \mathcal{V}_{q_{0,2}}$ as:
$\mathcal{V}_{\langle q_1, q_2 \rangle}(m)$ is $\mathcal{V}_{q_1}(m)$ if $m \in \mathcal{M}_1$, and if $m \in \mathcal{M}_2$ it will be $\mathcal{V}_{q_2}(m)$.

3 Overview and Formal Modeling of WS-CDL

The primary goal of the WS-CDL specification is to specify a declarative, XML-based language that defines from a global viewpoint the common and complementary observable behavior of participants, specifically the information exchanges that occur and the jointly agreed ordering rules that need to be satisfied. WS-CDL describes interoperable, peer-to-peer collaborations between participants without assuming single point of control. As mentioned before, WS-CDL is a description and not an executable language. Each participant, adhering to a WS-CDL collaboration representation, could be implemented using completely different mechanisms such as executable business process languages, e.g. BPEL, programming languages, e.g. C#, or human controlled software agents. In order to use WS-CDL for design time or static validation and verification of choreographies and conformance validation it must be based on or related to a formal language. In this section, first we define CASM$^{\text{ch}}$, a variant form of CASM customized for modeling choreographies, and afterwards we go through modeling each constituent part of WS-CDL by Reo and CASM$^{\text{ch}}$.

3.1 Constraint Automata with State Memory for Choreography

A choreography can be recognized as a container for a collection of activities that may be performed by the participants. In our model, each participant is considered as a Reo component. In order to make the CASM proper for modeling activities, we impose a convention on its name set to include two parts: *component name* and *port name* instead of just one part, *port name*. On the other hand, in WS-CDL, variables which are used for holding information and values of states are uniquely identified by the participants they belong to. As we model variables by a Reo component with one memory cell for holding its value, the name of the memory cells and the variable component are specified by two parts: *component name* and *variable name* in which the component name stands for the participant name the variable belongs to. Each part is separated from the other parts by ".".

Definition 3.1 (*Constraint Automata with State Memory for Choreography (CASM^{ch})*). A CASM$^{\mathrm{ch}}$ is a tuple $\mathcal{A}^{ch} = (Q^{ch}, \mathcal{N}ames^{ch}, \rightarrow^{ch}, Q_0^{ch}, \mathcal{M}^{ch}, \mathcal{V}_0^{ch})$ in which:

- Q^{ch}, Q_0^{ch}, $\rightarrow^{ch}$, and $\mathcal{V}_0^{ch}$ are defined the same as states, initial states, transition relation, and initial value function in ordinary CASM.
- Each $n \in \mathcal{N}ames^{ch}$, is a tuple (*component name*, *port name*).
- Each $m \in \mathcal{M}^{ch}$ is a tuple (*component name*, *variable name*).

The definition for the product of two CASM$^{\mathrm{ch}}$ is the same as for ordinary CASM by taking into account that during product stages two names or memory cells will match iff all of their constituent parts are equal.

3.2 Formalizing WS-CDL

Formally, a complete WS-CDL model is described by a set of *choreographies*. A WS-CDL choreography description is essentially a container for a collection of activities that may be performed by one or more of the participants and consists of three parts: choreography life-line, choreography exception blocks and choreography finalizer blocks. The choreography life-line expresses the progression of a collaboration through enabled activities and enclosed choreographies. Activities describe the actions performed within a choreography. The basic building block of a choreography is the interaction activity which results in an exchange of information between participants while the control-flow among activities is described by ordering structures activities.

We model each WS-CDL model as a Reo circuit. In our model, the WS-CDL participants are represented as Reo black-box components in the Reo circuit of WS-CDL. Instead of having a "process view" to the choreography, we extract and visualize the communication (interaction activities) and coordination (ordering structures activities) among the participants. All the data (variables) used for control decisions are extracted and shown in the coordination Reo circuit in between the participants. This Reo circuit can now be considered to be exogenously coordinating the participants. Each part of WS-CDL which is modeled by a Reo component obeys from a consistent pattern that contains input port *Start* and output ports *Ex* (for exception throwing) and *End*. In the following, we present our modeling of WS-CDL with more details.

3.2.1 RoleType, relationshipType and participantType

Within WS-CDL, a *participantType* groups together those parts of the observable behavior of a participant that must be implemented by the same logical entity or abstract organization. A *roleType* enumerates potential observable behavior a *participantType* can exhibit in order to interact. A *relationshipType* identifies the mutual commitments that must be made for collaborations to be successful. In our model, *participantTypes* can be considered as components in a Reo circuit, a *roleType* is a group of ports which implements its observable behaviors (each behavior can be implemented by a port in components) and the *relationship type* will be the connections in Reo circuit which determine which components can interact with each other.

3.2.2 InformationType and variable

A variable contains information about commonly observable objects in a collaboration, such as the information exchanged or the observable information of the *roleTypes* involved which have *informationTypes* that define the type of information the variable contains. In our mapping, variables are modeled as a Reo component with one memory cell in its CASM^{ch} which is shown in Fig. A.1.a of Appendix.

3.2.3 Choreography

A choreography defines collaborations between interacting *participantTypes*. The choreography life-line expresses the progression of a collaboration. An *exception block* specifies what additional actions should occur when a choreography behaves in an abnormal way. A *finalizer block* specifies additional actions that should occur to modify the effect of an earlier successfully completed choreography, for example to confirm or undo the effect. In our model, the behavior of the Reo circuit as a whole describes the choreography. The ports *Start*, *End* and *Ex* are derived for the whole choreography life-line from each activity within it. As shown in Fig. A.1.b of Appendix the *End* and *Ex* output ports are connected to the *Start* input port of finalizer and exception blocks respectively to enable them.

3.2.4 ChannelType

A channel realizes a point of collaboration between participantTypes by specifying where and how information is exchanged. Within WS-CDL, channels are abstractly modeled as channelTypes. As channels in WS-CDL may be one-way or two-way, and in Reo each channel can only have two ends, we modeled each WS-CDL one-way channel by a *Sync* channel in Reo and the WS-CDL two-way channel is modeled by combining two one-way channels.

3.2.5 Activities and ordering structures

Activities describe the actions performed within a choreography. Ordering structures combine activities with other ordering structures in a nested structure to express the ordering rules of actions performed within a choreography. In our model, each basic activity is modeled by a component whose behavior is specified by a $CASM^{ch}$. The ordering structures of activities (control flow activities) are modeled and visualized by Reo circuits coordinating the activities.

Basic Activities

- *Interaction* activity is the basic building block of a choreography. It results in an exchange of information between participants and possible synchronization of their observable information changes. Each interaction is composed of: (i) the participant roles involved; (ii) the information exchange type and the corresponding direction(s) which is described by the channels; (iii) the observable information changes. Like channel type, an interaction can be one-way (request or response) or two-way (request-response). The WS-CDL code of an example of one-way interaction in which through channel "ch" role X sends its request whose content is in variable "a" to the role Y and Y saves the received request message in its variable "b" is presented in the following:

```
<interaction name="sendRequest" channelVariable="ch"> <participate ...
                                                    "fromRoleTypeRef="X" toRoleTypeRef="Y"/>
    <exchange name="Request_a" ... action="request">
       <send variable="cdl:getVariable('a','','')"/>
       <receive variable="cdl:getVariable('b','','')"/>
    </exchange>
    </interaction>
```

  Its corresponding Reo is illustrated in Fig. A.1.c of Appendix. It must be noted that in a Reo component that is representative of a WS-CDL participant, each port must be uniquely identified by the information exchanged and the WS-CDL channel name through which the information is exchanged. This convention is used in the CASM$^{\text{ch}}$ shown in Fig. A.1.c of Appendix, e.g. *X.send(a)ch*. For two-way interaction via channel "ch", as shown in Fig. A.1.d of Appendix, two one-way channels are composed and the resulted Reo circuit is identified by "ch" as its name. In this figure, we abstracted away from the variables in each role and the internal view of each one-way interaction (shown as components). The resulted circuit as a whole is a component whose behavior is shown by CASM$^{\text{ch}}$ in the Fig. A.1.d of Appendix. The *Start*, *End* and *Ex* ports are constructed from the two one-way components similar to the manner constructed from the two variables in the Fig. A.1.c of Appendix (in these figures, In1 and In2 stand for one-way and two-way interaction Reo components respectively).

- *Assign* activity assigns the value of one variable or expression to another variable within one roleType. For example, the assignment of variable a to variable a' of role X is depicted in Fig. A.1.e of Appendix.

- *Workunit* describes the conditional and, possibly, repeated execution of an activity. Syntactically, a Workunit activity has several parts, including a reference to the enclosed activity, a guard (G), a block condition (*block*), and a repetition condition (*!R*). While G and *!R* conditions direct the repetition of the execution, the block condition determines whether to wait for G to become true or not. The modeling of non-blocking workunit ($block$ = "false") is shown in Fig. A.1.f of Appendix. For the blocking workunit the model is similar.

Ordering Structures

Ordering structures combine activities with other ordering structures in a nested structure to express the ordering rules of actions performed within a choreography. There are three types of ordering structures: *Sequence*, *Parallel*, and *Choice*. As illustrated in Fig. A.2 of Appendix, the *Sequence* is modeled by a Reo circuit, *Sequencer-n*, which enables its n connected components sequentially, the parallel execution of activities is handled by *replicator* node in Reo, and *X-Router-n* which is a Reo circuit that selects one active component among n connected components non-deterministically, is used for *Choice*.

4 End-Point Projection and Conformance Test

In order to investigate the conformance of a given party or orchestration to a choreography, end-point projection must be performed. In this operation, the desired behavior of a party (end-point process) is extracted from a choreography (global description). In our approach, each choreography can be modeled by a Reo circuit. In order to extract the behavioral interface of a choreography with respect to a certain participant, we shall hide all the nodes in the Reo circuit except the nodes of the desired component. Accordingly, the external behavior of a component can be extracted from the CA of a Reo circuit by applying the hide operation defined for CA. Therefore, as an advantage of our approach, we can use the already defined operator in our formal model, the hide operator, for end-point projection. After end-point projection, the conformance problem is reduced to comparison of the behavior of the given party (implementation) with the specification extracted by end-point projection. Again, the simulation relation defined for CA can be a satisfactory definition in this regard. Although, in comparison to CA, some changes in the definition of simulation relation and hide operator in CASM and CASM$^{\text{ch}}$ are needed.

4.1 End-point projection

In our approach, the behavior of each choreography is obtained in the form of CASM$^{\text{ch}}$ compositionally and the end-point projection can be implemented by the hide operation. So, we inspire from the hide operation for ordinary CA in [8] and define a new hide operation which is proper for CASM$^{\text{ch}}$. Before end-point projection, we assume that the examined choreography models are locally implementable according to the criteria mentioned in [25] which can be summarized as: 1) in a sequence, initiators (the participants which initiate the global activity) of the following activity must act as terminators (the participants which conclude the global activity) of the preceding activity, 2) in non-deterministic or conditional choice, each selective branch should have an identical initiator, and 3) in a workunit which has b as its loop condition each initiator of the enclosed activity has a corresponding loop condition embedded in b.

Definition 4.1 *End-point Projection.* The end-point projection is a function $\pi : CASM^{ch} \times Participant \to CASM$ where if $\mathcal{A} = (Q, \mathcal{N}ames, \to, Q_0, \mathcal{M}, \mathcal{V}_0)$ is a CASM$^{\text{ch}}$ and P is the *component name* part of at least one $n \in \mathcal{N}ames$, then the end-point projection of $\mathcal{A}$ on participant P is defined as $\pi(\mathcal{A}, P) = (Q_{\pi(P)}, \mathcal{N}ames_{\pi(P)}, \to_{\pi(P)}, Q_{0,\pi(P)}, \mathcal{M}_{\pi(P)}, \mathcal{V}_{0,\pi(P)})$ where:

- $\mathcal{N}ames_{\pi(P)}$ and $\mathcal{M}_{\pi(P)}$ stand for the projection of name set and memory cells on P. In other words, they consist of the set of names and memory cells whose *component name* part is P and the *component name* part are removed from their representations.
- $Q_{\pi(P)}$ is equal to the set of states Q, except that the value function of each $q \in Q_{\pi(P)}$ is defined as $\mathcal{V}_{q,\pi(P)} : \mathcal{M}_{q,\pi(P)} \to Data$ in which $\mathcal{M}_{q,\pi(P)}$ stands for the projection of memory cells $\mathcal{M}_q$ on participant P.
- For definition of the initial state, let $\to^*$ be a transition relation such that $p \to^* q$ iff there exists a finite path ($\sim P$ denotes the names whose *component name* part are not P) $p \xrightarrow{\sim P, g_1} q_1 \xrightarrow{\sim P, g_2} q_2 ... \xrightarrow{\sim P, g_n} q_n$ Where $q_n = q,\ g_1, g_2, ..., g_n$ are satisfiable, and the *component name* part of the memory cells used in each $(m, d) \in V_{qi},\ i \in \{1, .., n\}$ is not P, then the set of initial states will be:
  $Q_{0,\pi(P)} = \{p_{\pi(P)} | p \in Q, q_0 \to^* p\ for\ some\ q_0 \in Q_0\}$
- The transition relation $\to_{\pi(P)}$ is given by:

$$\frac{p \ \to^* q,\ q \xrightarrow{N,g} r,\ N_{\pi(P)} \neq \phi \vee \mathcal{V}_{r,\pi(P)} \neq \phi}{p_{\pi(P)} \ \xrightarrow{N_{\pi(P)}, g_{\pi(P)}}_{\pi(P)} \ r_{\pi(P)}}$$

  in which $g_{\pi(P)}$ is the projection of data constraint g on P which is obtained by removing the data constraints whose *component name* parts are other than P and replacing them with true or false.

4.2 Conformance analysis

According to [28] each orchestration described by BPEL can be specified by a Reo circuit and its corresponding CA. In our Reo circuit extracted from a BPEL code, we can also hide the internal behavior and only keep the observable behavior (abstract BPEL process). Since each CA can be considered as a special case of CASM, we can have the behavior of each orchestration in the form of a CASM. Thanks to this unified formalism, after end-point projection, the conformance problem is reduced to the examination of the CASM obtained by end-point projection and the CASM of the orchestration.

Intuitively, conformance is the capability of a web service to interact with the other peers according to a choreography. A web service will be conformant to a choreography if it complies to the choreography in all possible interactions [17]. This intuitive definition is not directly usable for automatic verification as there is no unique formal definition for conformance. Therefore, the relationships between choreography and behavioral interface (called abstract process in WS-BPEL) of web services may be non-trivial, and there are currently no precise notions of conformance between WS-CDL choreographies and WS-BPEL abstract processes [10]. Understanding these relations is considered as an open problem in [10]. Different works assume different criteria for the conformance and formalize it according to their domain of usage. These criteria are generated by focusing on different aspects of behaviors of web services. Most of the works consider inclusion or equivalence relation between the set of possible actions of an orchestration and what is desired by the choreography, some works distinguish between incoming and outgoing messages, and some take into account the termination patterns and/or causal relationship between messages.

The work in [9] uses automata for modeling choreographies and formalizes the conformance regarding to three criteria: first, it guarantees that the service, at any point of its conversations, can only send messages which are legal w.r.t. the global interaction protocol. Moreover it guarantees that the service will be able to handle any incoming message, foreseen by the protocol, and finally, it guarantees that the service will always send at least one of the messages foreseen by the protocol, although it is not necessary that its policy envisions all the possible alternatives (e.g. the designer can restrict the set of the possible answers). In [24] the conformance is defined by process refinement. A process P is a refinement of process Q, if P cannot perform an infinite sequence of internal actions and also whichever observable action that process P can do is prescribed by process Q, whereas the internal actions do not count. Given a locally implementable choreography A and an orchestration P which acts as a participant ρ, it assumes P is conformant to A if and only if P is a process refinement of the process resulted by end-point projection of A on ρ. In [11] the criteria for checking the conformance of web service contracts and business processes are defined as: 1) all incoming messages specified in contract must be handled by the business process and vice versa, 2) the business process must handle all outgoing messages specified in its contract and the service contract must handle all outgoing events in business process, 3) the causal relationships between incoming and outgoing messages must be maintained when generating business process from service contract and vice versa, and 4) the service contract must reflect all acceptable termination states of a business process and vice versa. Similarly, the termination patterns in both business processes and service contracts must match. The authors in [17] assume a web service is compliant if all the interactions it can possibly generate belong to the choreography specification. Similar definition can be seen in [13] in which it is assumed that it is not necessary for an implementation to include all possible conversations admitted by a choreography but all possible conversations in an implementation must be admitted by the choreography.

In CA, the input and output actions and also termination and deadlock states are not distinguished. On the other hand, as mentioned before, we abstract away from internal actions of a web service by hide operation and use abstract BPEL processes. After hiding, we have no information about the internal actions of web services and the casual relationship between messages. In our conformance definition, we assume a web service conforms to a choreography if every action allowed by the web service is also allowed by the behavioral interfaces extracted from choreography by end-point projection, i.e. an orchestration that performs actions which are not foreseen and desired by the choreography, may disturb the communication of web services. Hence, there should be no action in the web service that is not foreseen by the choreography. This is obtained when in each state all possible actions of the web service (all possible transitions in the CASM that can be traversed) are a subset of the possible actions desired by the choreography. In other words, conformance is obtained when the satisfaction of the data constraints in each transition in each state of the web service implies the satisfaction of the data constraint of the corresponding transition in the CASM of the behavioral interface. This examination can be implemented by utilizing the predefined concept in ordinary CA [8], the *simulation relation*. But here, we need to customize the simulation definition of ordinary CA with respect to the notions in CASM. In the following, we define the simulation relation in CASM and show how to exploit that for conformance validation.

Definition 4.2 *Simulation Relation in CASM.* Given two CASM $\mathcal{A}_1 = (Q_1, \mathcal{N}ames, \rightarrow_1, Q_{0,1}, \mathcal{M}_1, \mathcal{V}_{0,1})$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}ames, \rightarrow_2, Q_{0,2}, \mathcal{M}_2, \mathcal{V}_{0,2})$:

(i) The binary relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is a simulation between $\mathcal{A}_1$ and $\mathcal{A}_2$ if and only if for all $q_1 \in Q_1$ and $q_2 \in Q_2$, if $(q_1, q_2) \in \mathcal{R}$, then the following conditions hold:
  (a) $\mathcal{V}_{q_2} \subseteq \mathcal{V}_{q_1}$.
  (b) for each pair of states $(q_1, p_1) \in\rightarrow_1$ and every $n \subseteq \mathcal{N}ames$, there exists a pair $(q_2, p_2) \in\rightarrow_2$ such that: $dc(q_1, n, p_1) \leq dc(q_2, n, p_2)$ and $(p_1, p_2) \in \mathcal{R}$.
  Where $dc(q, n, p)$ denotes the data constraint of a transition from state q to state p with the name set n as defined in [8] and the $\leq$ relation means that satisfaction of the left hand side constraint implies the satisfaction of the right hand side constraint. It must be noted that if there is no transition from state q to state p with the name set n, then $dc(q, n, p) = false$. A state q_1 is simulated by another state q_2 (i.e., q_2 simulates q_1), denoted as $q_1 \preceq q_2$, iff there exists a simulation $\mathcal{R}$ with $(q_1, q_2) \in \mathcal{R}$.

(ii) We say that $\mathcal{A}_2$ simulates $\mathcal{A}_1$ (denoted as $\mathcal{A}_1 \preceq \mathcal{A}_2$), iff $\mathcal{M}_2 \subseteq \mathcal{M}_1$ and every initial state of $\mathcal{A}_1$ is simulated by an initial state of $\mathcal{A}_2$.

Similar to the simulation relation in ordinary CA, we rely on the same set of names for both automata which corresponds to the I/O ports of Reo components. This is desirable for us, because existence of extra ports for the orchestration may disturb the interoperability of participants.

Definition 4.3 *Conformance.* An orchestration whose external behavior is defined by a CASM $\mathcal{A}_{orch}$, is conformant to a choreography with CASM$^{\text{ch}}\mathcal{A}_{ch}$, for playing the role of participant P, if the $\pi(\mathcal{A}_{ch}, P)$ simulates $\mathcal{A}_{orch}$ or $\mathcal{A}_{orch} \preceq \pi(\mathcal{A}_{ch}, P)$.

5 Case Study: The Bartering Protocol

Now as an example, we consider bartering protocol used in [4,25] with some changes. In this system, the buyer repeatedly asks for a quote from seller until he agrees with the quote and places an order. For this purpose, a *workunit* is used in which there is a condition that governs the repetition, namely `barteringDone = false` and `accept = false`. For the sake of simplicity, instead of its WS-CDL code, its pseudo-code is shown in the following (`a@R` means variable `a` of Role `R` [4]):

```
Boolean barteringDone@Seller = false
Boolean accept@Buyer = false
Elicit a quote from the seller

while (barteringDone@Seller == false and accept@Buyer == false) do
{
    choice
    {
        {
            Accept the quote
            accept@Buyer = true
            Place the order
            barteringDone@Seller = true
        }
        {
            Reject the quote and ask for a new quote
        }
    }
} done
```

The Reo circuit and CASM$^{\mathrm{ch}}$ of this system is shown in Fig. 2. In this figure, we abstract away from exception signals as there is no exception thrown in the example. The name of each component in the Reo circuit is the representative of the interaction performed within it. In addition, in each one of them some proper variable assignments are also performed as shown in the CASM$^{\mathrm{ch}}$. That is the advantage of Reo that enables us to encapsulate multiple activities in a component during visualization, while the details can be remained in its operational semantics for accurate analysis. In this CASM$^{\mathrm{ch}}$, `B` and `S` stand for "Buyer" and "Seller" participants respectively. The additional point is that as we abstract away from variables, for the sake of simplicity, the input port G of the workunit is not shown as its values come from the variables in the participants (note that we don't have $!R$ condition in the workunit).

In order to show the applicability of our proposed end-point projection and conformance analysis, consider a web service (orchestration) whose behavior is shown in Fig. 3. Our goal is to investigate whether it conforms to the bartering protocol for playing the role of participant "Buyer" or not.

The first step is to achieve the desired behavior of the "Buyer" by end-point projection which is shown in Fig. 4. The next step is the examination of conformance. As shown in Fig. 3, the automaton of the orchestration is similar to the automaton of Fig. 4; the only difference can be observed in states 5 and 5' where the decision for acceptance or rejection of the quote in the orchestration depends on the quote value ($quote < 100$ or $quote \geq 100$) instead of non-deterministic selection. In these states, we have $dc(5', accept.Write, 6') \leq dc(5, accept.Write, 6)$, $dc(5', send.(qupd)cu, 7') \leq dc(5, send(qupd)cu, 7)$, and $\mathcal{V}_5 \subseteq \mathcal{V}_{5'}$. In addition, we have $\{(6', 6), (7', 7)\} \subset \mathcal{R}$. Therefore, according to Definition 4.2, the orchestra-

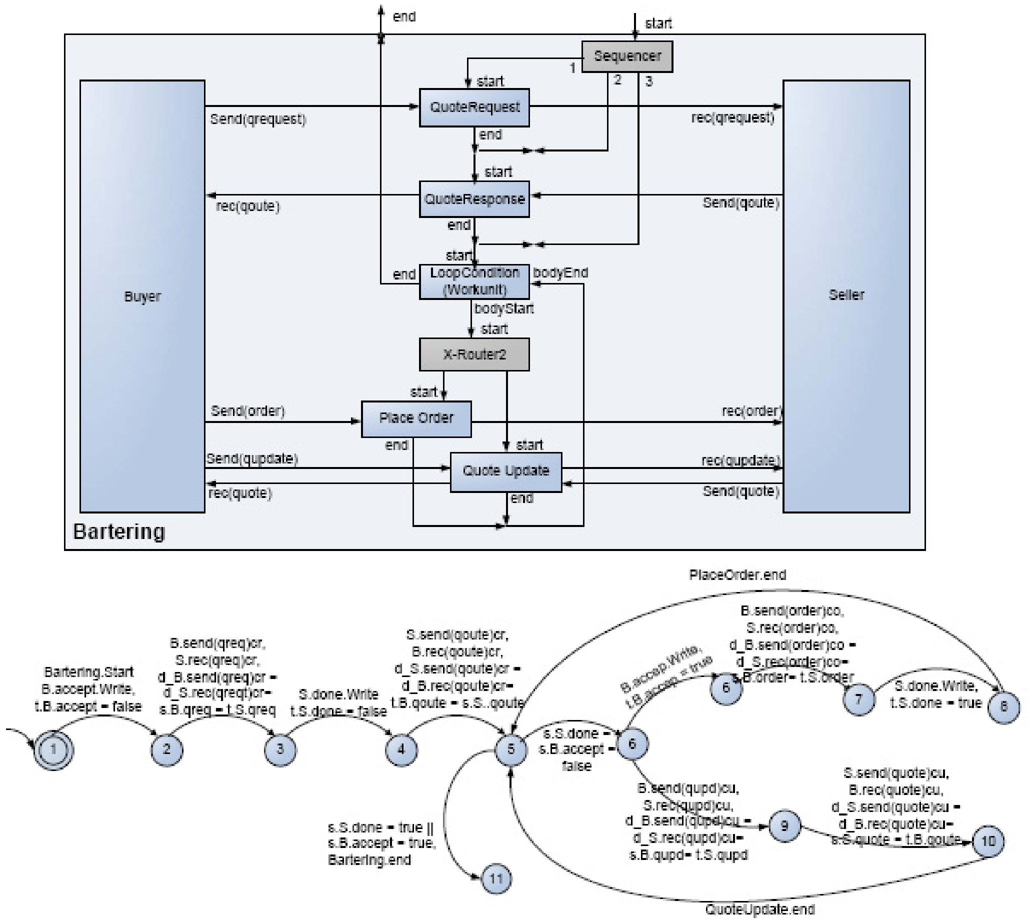
end
start
Sequencer
1
2
3
start
QuoteRequest
Send(qrequest)
rec(qrequest)
end
start
QuoteResponse
rec(qoute)
Send(qoute)
end
start
end
LoopCondition (Workunit)
bodyEnd
bodyStart
start
X-Router2
Buyer
Seller
start
Place Order
Send(order)
rec(order)
end
start
Quote Update
Send(qupdate)
rec(qupdate)
rec(quote)
Send(quote)
end
Bartering
Bartering.Start B.accept.Write, t.B.accept = false
B.send(qreq)cr, S.rec(qreq)cr, d_B.send(qreq)cr = d_S.rec(qreqt)cr= s.B.qreq = t.S.qreq
S.done.Write t.S.done = false
S.send(qoute)cr, B.rec(qoute)cr, d_S.send(qoute)cr = d_B.rec(qoute)cr= t.B.qoute = s.S..qoute
PlaceOrder.end
B.accep.Write, t.B.accep = true
B.send(order)co, S.rec(order)co, d_B.send(order)co = d_S.rec(order)co= s.B.order= t.S.order
S.done.Write, t.S.done = true
s.S.done = s.B.accept = false
B.send(qupd)cu, S.rec(qupd)cu, d_B.send(qupd)cu = d_S.rec(qupd)cu= s.B.qupd= t.S.qupd
S.send(quote)cu, B.rec(quote)cu, d_S.send(quote)cu = d_B.rec(quote)cu= s.S.quote = t.B.qoute
s.S.done = true || s.B.accept = true, Bartering.end
QuoteUpdate.end
1
2
3
4
5
6
6
7
8
9
10
11

Fig. 2. The Reo circuit and CASM$^{\text{ch}}$ of the bartering protocol

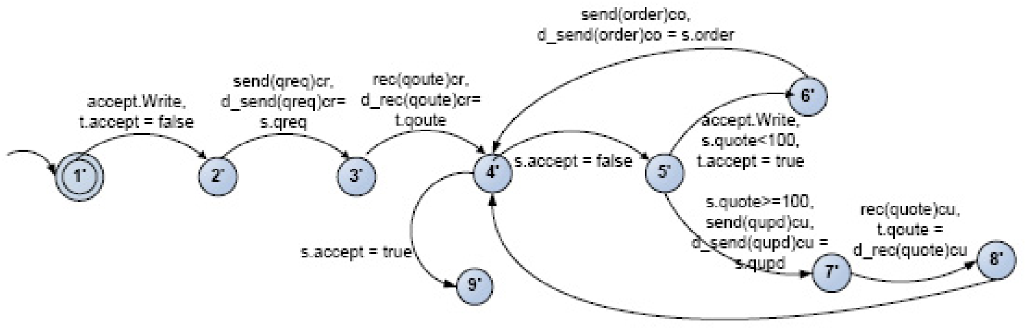
send(order)co, d_send(order)co = s.order
accept.Write, t.accept = false
send(qreq)cr, d_send(qreq)cr= s.qreq
rec(qoute)cr, d_rec(qoute)cr= t.qoute
s.accept = false
accept.Write, s.quote<100, t.accept = true
s.quote>=100, send(qupd)cu, d_send(qupd)cu = s.qupd
rec(quote)cu, t.qoute = d_rec(quote)cu
s.accept = true
1'
2'
3'
4'
5'
6'
7'
8'
9'

Fig. 3. The CASM of the "Buyer" web service

tion conforms to the protocol as its initial state is simulated by the initial state of the automaton resulted from end-point projection, using the simulation relation $\mathcal{R} = \{(1', 1), (2', 2), (3', 3), (4', 4), (5', 5), (6', 6), (7', 7), (8', 8), (9', 9)\}$.

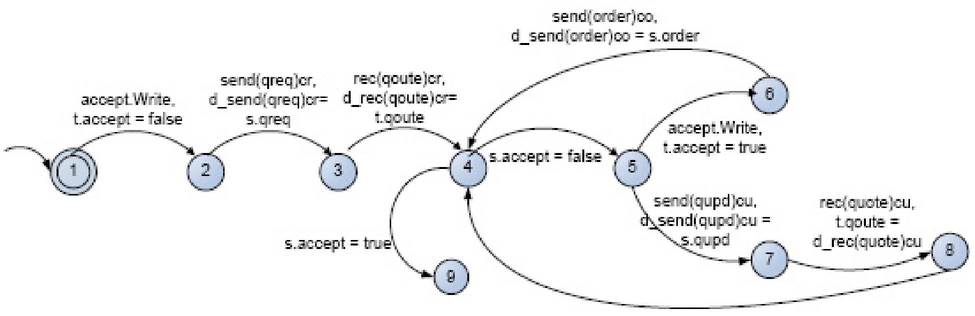

Fig. 4. The end-point projection of the bartering protocol on the "Buyer" participant

6 Related Work

The works on formal specifications of choreographies and specially WS-CDL, aims towards addressing some important challenges as: (1) design time or static validation and verification of choreographies; (2) automatic generation of local implementations from a given global model (interaction protocol); (3) verification of the conformance of a given process behavior to a general interaction protocol

Busi et al [15] design a simple choreography language CL whose main concepts are based on WS-CDL. This language covers a few constructs, nevertheless it offers a starting point for the design and analysis of choreography. Gorrieri and others [21] give a deep analysis of interaction patterns in the WS-CDL specification by taking into account the alignment property, whose meaning is related to the possibility to control when the interaction completes. The work in [29] proposes a small language, CDL, and its operational semantics as a formal model of simplified WS-CDL. Verification of some properties are also presented in a choreography sample using SPIN model-checker. A more complete work on formal specification of WS-CDL can be found in [24] in which the semantics of WS-CDL is presented in terms of process algebra CSP.

Among the works on projection and conformance analysis, Busi et al. [14,16] formalize choreography and orchestration using process algebra, where conformance takes the form of a bisimulation-like relation. By means of automaton, the work in [9] defines a conformance notion which tests whether interoperability is guaranteed. In [20] a conversation protocol is specified by a realizable Büchi automaton, and the peer implementations are synthesized from the protocol via projection. Zhao et al. [30] propose a small language as a formal model of the simplified WS-CDL and projected a given choreography to orchestration views. In [19] the authors use Petri nets for describing orchestration, choreography and service interface behavior focusing on the relationship between a single orchestrator with respect to a given choreography. The work in [25] introduces two languages for describing choreography and orchestration and based on them gives a definition of endpoint projection. The interesting point with this paper is formalizing three conditions that need to be satisfied so that a global model can be locally implementable. The authors in [27] use Reo and CA to investigate the issues of description, orchestration, and choreography of web services at a unifying abstract level. In this work, only limited notions of WS-CDL are studied and modeled. Moreover, the conformance and projection issues are not addressed.

The advantages of our work to these works are integrating the following issues: (1) taking into account a specific choreography language which makes our work more useful in practice; (2) covering almost all of the important notions in WS-CDL such as activities, channels, exception handling, finalizer blocks, etc.; (3) using Reo for visualizing models in different abstract levels and also hierarchical composition. As Reo is proposed as a coordination language for coordination of concurrent components, its computational model inherently fits for coordination and communication of web services as self contained components. Furthermore, in comparison to other formal languages, e.g. process algebra, which are just represented by mathematical relation, visual modeling of data flow among components in Reo makes the comprehension of component interaction more intuitive. On the other hand, thanks to hide operation and consequently hierarchical composition in Reo, the connectors can be specified in any abstract level and so the modeling can be more comprehensive and better matched with its real configuration; (4) formalizing end-point projection and conformance issues. In other words, this work which is a complement of our previous work [28], in which we modeled BPEL language by Reo and CA, gives us a unified formalism which facilitates verification and validation of web service specification, implementation and conformance in a consistent manner.

7 Conclusion and Future Work

In this work, we formalize the choreography description language, WS-CDL, using Reo and CASM. By presenting a slightly modified version of CASM, CASM$^{\mathrm{ch}}$, and translating each component of WS-CDL to its corresponding Reo and CASM$^{\mathrm{ch}}$, the behavior of a choreography will be obtained in the form of CASM$^{\mathrm{ch}}$. We define the end-point projection operator on CASM$^{\mathrm{ch}}$ whose outcome is in the form of CASM. Regarding to our previous work in which the BPEL orchestration language is modeled by Reo and CA (the simplified form of CASM), both of the orchestration and end-point projected choreography can be presented with a unified formalism through which conformance problem is resolved by defining the simulation relation. The practical application of our work is illustrated in a bartering protocol.

In future, we are intended to work on the different aspects of conformance problem such as situations that make a choreography, specified in terms of Reo and CASM$^{\mathrm{ch}}$, locally implementable. In our proposed model for WS-CDL, all of the decisions that control the flow of events are extracted and visualized by the Reo circuit in between participants, i.e., the coordination is performed exogenously. This can facilitate discovering the choreographies that are not locally implementable. Additionally, developing a tool which automates the translation from WS-CDL to Reo and CASM$^{\mathrm{ch}}$and conformance analysis constitutes our future plans. This tool must be integrated with the tools that translate WS-BPEL to Reo [26], Reo to CA, and CA verification tool [23]. Our work can also be extended to analyze the power and weakness of WS-CDL as a choreography description language and, if needed, adding proper concepts to WS-CDL, Reo, and CASM$^{\mathrm{ch}}$ to make them more suitable for choreography description and modeling. This can be done in conjunction with property specification and analysis of choreographies.

References

[1] *Business Process Modeling Language BPML* (2002), http://www.bpmi.org.

[2] *Web services choreography requirements*, W3C Working Draft (2004).

[3] *Web services choreography description language version 1.0*, W3C Candidate Recommendation (2005), http://www.w3.org/TR/ws-cdl-10/.

[4] *Web services choreography description language: Primer*, W3C Working Draft (2006).

[5] Arbab, F., *Reo: A channel-based coordination model for component composition*, Mathematical Structures in Computer Science **14** (2004), pp. 329–366.

[6] Arbab, F., *Constraint automata with state memory*, Unpublished notes (2007), CWI.

[7] Arbab, F. and J. Rutten, *A coinductive calculus of component connectors*, in: M. Wirsing, D. Pattinson and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques, Proceedings of 16th International Workshop on Algebraic Development Techniques (WADT 2002)*, Lecture Notes in Computer Science **2755**, 2003, pp. 35–56, http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R0216.pdf.

[8] Baier, C., M. Sirjani, F. Arbab and J. Rutten, *Modeling component connectors in Reo by constraint automata*, Science of Computer Programming **61** (2006), pp. 75–113.

[9] Baldoni, M., C. Baroglio, A. Martelli, V. Patti and C. Schifanella, *Verifying the conformance of web services to global interaction protocols: A first step*, in: Bravetti et al. [12], pp. 257–271.

[10] Barros, A., M. Dumas and P. Oaks, *A critical overview of web service choreography description language(WS-CDL)* (2005), BPTrends.

[11] Bhuiyan, J., S. Nepal and J. Zic, *Checking conformance between business processes and web service contract in service oriented applications*, in: *ASWEC '06: Proceedings of the Australian Software Engineering Conference* (2006), pp. 80–89.

[12] Bravetti, M., L. Kloul and G. Zavattaro, editors, "Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings," Lecture Notes in Computer Science **3670**, Springer, 2005.

[13] Bravetti, M. and G. Zavattaro, *Towards a unifying theory for choreography conformance and contract compliance*, in: *Proceeding of Software Composition 2007*, LNCS **4829** (2007), pp. 34–50.

[14] Busi, N., R. Gorrieri, C. Guidi, R. Lucchi and G. Zavattaro, *Choreography and orchestration: A synergic approach for system design*, in: B. Benatallah, F. Casati and P. Traverso, editors, *ICSOC*, Lecture Notes in Computer Science **3826** (2005), pp. 228–240.

[15] Busi, N., R. Gorrieri, C. Guidi, R. Lucchi and G. Zavattaro, *Towards a formal framework for choreography*, in: *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise* (2005), pp. 107–112.

[16] Busi, N., R. Gorrieri, C. Guidi, R. Lucchi and G. Zavattaro, *Choreography and orchestration conformance for system design*, in: P. Ciancarini and H. Wiklicky, editors, *COORDINATION*, Lecture Notes in Computer Science **4038** (2006), pp. 63–81.

[17] Chesani, F., P. Mello, M. Montali, M. Alberti, M. Gavanelli, E. Lamma and S. Storari, *Abduction for specifying and verifying web service and choreographies*, in: *4th International Workshop on AI for Service Composition*, Trento, Italy, 2006.

[18] Diakov, N. and F. Arbab, *Compositional construction of web services using Reo*, in: *Proceedings of International Workshop on Web Services: Modeling, Architecture and Infrastructure*, 2004, pp. 49–58.

[19] Dijkman, R. and M. Dumas, *Service-oriented design: A multi-viewpoint approach*, International Journal of Cooperative Information Systems **13** (2004), pp. 337–378.

[20] Fu, X., T. Bultan and J. Su, *Conversation protocols: a formalism for specification and verification of reactive electronic services*, Theor. Comput. Sci. **328** (2004), pp. 19–37.

[21] Gorrieri, R., C. Guidi and R. Lucchi, *Reasoning about interaction patterns in choreography*, in: Bravetti et al. [12], pp. 333–348.

[22] Jordan, D. and J. Evdemon, *Web Services Business Process Execution Language Version 2.0* (2006).

[23] Klüppelholz, S. and C. Baier, *Symbolic model checking for channel-based component connectors*, in: *FOCLASA'06*, 2006.

[24] Li, J., J. He, H. Zhu and G. Pu, *Modeling and verifying web services choreography using process algebra*, in: *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, 2007, pp. 256–268.

[25] Li, J., H. Zhu and G. Pu, *Conformance validation between choreography and orchestration*, in: *TASE '07: Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering* (2007), pp. 473–482.

[26] Mahdikhani, F., *BPEL to Reo tool* (2007), `http://ece.ut.ac.ir/msirjani/B2ReoTool/B2R.jar`.

[27] Meng, S. and F. Arbab, *Web services choreography and orchestration in Reo and constraint automata*, in: *Proceedings of 22nd Annual ACM Symposium on Applied Computing(SAC'07)*, 2007.

[28] Tasharofi, S., M. Vakilian, R. Zilouchian and M. Sirjani, *Modeling web service interactions using the coordination language Reo*, in: *Proceeding of 4th International Workshop on Web Services and Formal Methods (WS-FM 2007)*, LNCS **4937** (2008), pp. 108–123.

[29] Yang, H., X. Zhao, Z. Qiu, G. Pu and S. Wang, *A formal model for web service choreography description language (ws-cdl)*, in: *ICWS '06: Proceedings of the IEEE International Conference on Web Services* (2006), pp. 893–894.

[30] Zhao, X., H. Yang and Z. Qiu, *Towards the formal model and verification of web service choreography description language*, in: M. Bravetti, M. Núñez and G. Zavattaro, editors, *WS-FM*, Lecture Notes in Computer Science **4184** (2006), pp. 273–287.

A Modeling of WS-CDL components by Reo and CASM^{ch}

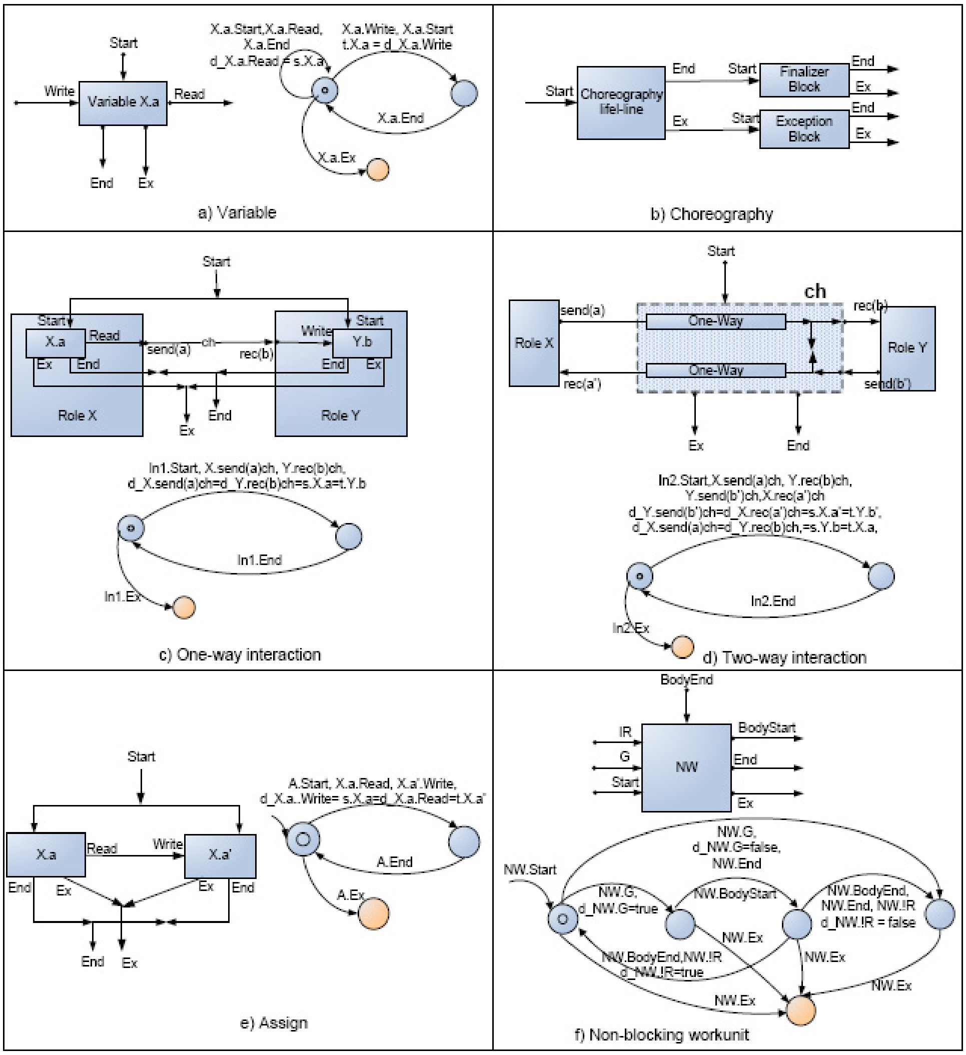

Fig. A.1. Modeling of variable, choreography, and basic activities

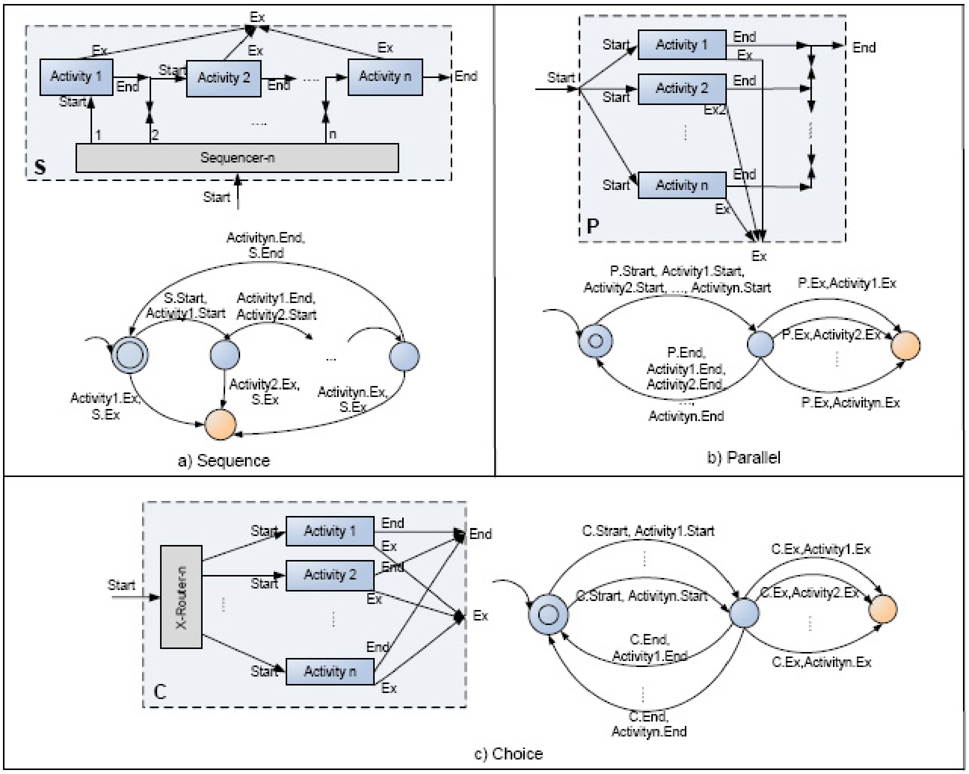

Fig. A.2. Modeling of ordering structures