# Reasoning about interaction patterns in Choreography

Roberto Gorrieri, Claudio Guidi, and Roberto Lucchi

Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
e-mail:{gorrieri,cguidi,lucchi}@cs.unibo.it

**Abstract.** Choreography languages provide a top-view design way for describing complex systems composed of services distributed over the network. The basic building block of such languages is the interaction between two peers which are of two kinds: request and request-respond. WS-CDL, which is the most representative choreography language, supports a pattern for programming the request interaction and two patterns for the request-respond one. Furthermore, it allows to specify if an interaction is aligned or not whose meaning is related to the possibility to control when the interaction completes. In this paper we reason about interaction patterns by analyzing their adequacy when considering the fact that they have to support the alignment property. We show the inadequacy of the two patterns supporting the request-respond interaction; one of them because it does not permit to reason on alignment at the right granularity level and the other one for some expressiveness lacks.

## 1 Introduction

Service Oriented Computing (SOC) paradigm provides a mean to design complex systems by exploiting and composing services available over the network. Web services technology, which is one of the most prominent technologies for SOC, provides several languages for composing services, the so-called orchestration (e.g., WSFL [4], XLANG [6], WS-BPEL [5]) and choreography (e.g. WS-CDL [7], WSCI [10]) languages. Althought there is not a common agreement on the meaning of orchestration and choreography, in [1] we have shown how orchestration and choreography work at different levels. On the one hand orchestration describes how to compose services from the point of view of a single entity, the so-called orchestrator, which coordinates the entire system, while on the other hand, choreography describes all the interdependencies among the different interactions between participants in a top-view way.

The most interesting proposal for choreography is Web Services Choreography Description Language (WS-CDL) which is an XML-based language. In a few words, choreography is composed of a static part describing the system (i.e. participants, variables and channels) and another one describing the behavior, that is the conversation rules of the system. The basic building block of WS-CDL for describing the conversational part is the interaction that can be composed

by using sequential, parallel and alternative operators. An interaction describes a messages exchange between two participants and can be of two kinds: *request* and *request-respond*. A request interaction consists of a message exchanged by a participant to another one, while a request-respond is composed of a message transmitted by a participant to another one followed by a response message in the opposite direction.

Since the request-respond is a complex interaction containing two message exchanges, it is possible to define it in different ways. Indeed, it is possible to program separately the request and the response or programming them with a construct dealing with the entire request-respond interaction in an atomic way. Here we talk about interaction patterns referring to the language mechanisms which allow to express the request and the request-respond interaction types.

WS-CDL supports three different interaction patterns: the *request*, the *atomic request-respond* and the *splitted request-respond*. The request pattern simply maps the request interactions type. The splitted request-respond pattern allows to manage separately the request and the respond message exchanges, thus allowing for example to describe that a certain activity have to be performed after the request and before the respond. The atomic request-respond pattern allows to express the entire request-respond interaction as an atomic construct; in this case it is not possible to associate an activity that is to be performed "within" that interaction. An interesting property which can be associated to interactions is the *alignment*, that is the possibility to control when an interaction completes. WS-CDL permits to set alignment property at the level of both single message exchange (request and respond), by using the splitted request-respond pattern, and at the level of entire request-respond interaction by using the atomic one.

In this paper we reason about the interaction patterns analyzing their adequacy considering the fact that they have to support the alignment property. To this end we present a core sublanguage of WS-CDL, equipped with a formal semantics, supporting the three interaction patterns above. We show that for the request-respond interaction the splitted pattern does not capture the essence of the alignment property and it allows to program not valid conversations, while the atomic one allows to reason at the right granularity level but it lacks in expressiveness w.r.t. the splitted one. Furthermore, by using some examples we show that the atomic request-respond pattern has less expressive power than the splitted one. The investigation described above have one meaningful implication: the formal semantics of a significant fragment of WS-CDL has been defined. Indeed, to the best of our knowledge, this work represents the first attempt towards the formalization of WS-CDL semantics.

Finally, in light of the results obtained in this paper we show how our formal language $CL$ presented in [3] deals with the right granularity level for the request-respond interaction, concluding that it can be considered a good starting point towards the definition of a formal framework for choreography.

The paper is structured as follows. Section 2 presents a core language of WS-CDL and its semantics. Section 3 is devoted to formally reason on the

request-respond interaction patterns provided by WS-CDL. Section 4 concludes the paper with some final remarks.

## 2 The $WSCDL_{core}$ language

As previously mentioned WS-CDL is composed of a static part describing the system (i.e. participants, variables and channels) and another one describing the conversation rules the participants have to follow.

In [3] we have presented a formal language proposal for representing choreography where we have extracted its essence following the same description approach of WS-CDL. Our proposal indeed, is composed by two parts: a *declarative* part and a *conversational* one. The former deals with the formalization of the participants involved in the choreography whereas the latter deals with the conversation rules they have to follow in order to interact each other.

In this paper choreographies are defined by exploiting the declarative part of our formal model, while the conversation rules are expressed by introducing a formal language which accounts for a significant fragment of WS-CDL. It is worth noting that the declarative part is not a faithful representation of the WS-CDL contructs which deal with participants, variables and channels but it just addresses their basic concepts. However, since we are interested on reasoning about interaction patterns, we can abstract from these details.

As far as the conversational part is concerned, here we present the language $WSCDL_{core}$ which formally represents the WS-CDL patterns we are interested in. The semantics of this language will be presented in terms of another language $CL_P$. Although that there are few differences between them, this choice is due to the fact that it simplifies the comparison with our conversation language $CL$ whose semantics is defined in terms of the same language $CL_P$. In the following indeed, we are interested to test and verify the suitability of $CL$ considering the issues raised by this paper.

In the following we explain the declarative part and we present $WSCDL_{core}$ syntax and semantics.

### 2.1 Declarative part

Here we explain the declarative part of our choreography formal model which is based on the concept of *role*. A role represents the behaviour that a participant has to exhibit in order to fulfill the activity defined by the choreography. Each role can store variables and exhibit operations.

As far as variables are concerned, we associate to each role a set of variables which represent the information managed by the role and which will be used in the interactions between roles. In our proposal variables are only names without any values and they are exploited in order to represent the data flow among the roles.

As far as operations are concerned, each role is equipped with a set of operations it has to exhibit which essentially represent the access points that will be

used by the other roles to interact with the owner one. Operations can have one of the following interaction modalities: *One-Way* or *Request-Response*.

In particular we remind that operations are defined in WSDL specifications [9] and they are the only mechanism for interacting with a Web Service. Briefly, an operation contains the definition of an incoming message for a service and, when used, the definition of the response message. Two different kinds of operation exist:

- *One-Way*: only the incoming message is defined.
- *Request-Response*: both the incoming message and the response one are defined.

When a Web service is invoked by using a *One-Way* operation, it receives the incoming message and starts its activities. On the other hand, when a Web Service is invoked by using a *Request-Response* operation, the service receives the message, starts its activities and, at the end, replies to the invoker with a response message.

Let us now introduce the formalization of *roles*, *variables* and *operations*.

Let $Var$ be the set of variables ranged over by $x, y, z, k$. We denote with $\widetilde{x}$ tuples of variables, for instance, we may have $\widetilde{x} = \langle x_1, x_2, ..., x_n \rangle$.

Let $OpName$ be the set of operation names, ranged over by $o$, and $OpType = \{ow, rr\}$ be the set of operation types where $ow$ denotes a One-Way operation whereas $rr$ denotes the Request-Response one.

An operation is described by its operation name and operation type. Namely, let $Op$ be the set of operations defined as follows:

$$Op = \{(o, t) \mid o \in OpName,\ t \in OpType\}$$

A role is described by a role name, the set of operations it exhibits and by a set of variables. Namely, let $RName$ be the set of the role names, ranged over by $\rho$. The set $Role$, containing all the possible roles, is defined as follows:[1]

$$Role = \{(\rho, \omega, V) \mid \rho \in RName,\ \omega \in \wp(Op), V \in \mathbf{P}(Var)\}$$

Exploiting this formalization, in the following we present the $WSCDL_{core}$ language.

### 2.2 Conversational part

Before presenting the syntax of the $WSCDL_{core}$ language, we intend to spend some words in order to highlight some WS-CDL basics. In particular we focus on the basic building block `<interaction>` which can be composed exploiting the tags `<sequence>`, `<parallel>` and `<choice>` representing, respectively, the sequential, parallel and alternative composition. Now, we remind some basics of the WS-CDL `<interaction>` tag in order to explain its functioning.

[1] Given a set $S$, with $\wp(S)$ we denote the powerset of $S$.

The tag `<interaction>` is used for defining interactions and it has some enclosed elements and attributes. For the sake of this paper we are interested in discussing the inner tags `<participate>` and `<exchange>` and the attributes `channelVariables`, `operation` and `align`. We refer to WS-CDL specifications for more details.
The description of the inner tags follows:

- `<participate>`: this tag defines the participants involved in the interactions exploiting the attributes `relationshipType`, `fromRole` and `toRole`. In WS-CDL specification the participants involved in a choreography are described exploiting a complex hierarchical structure. Briefly, the roles express the observable behaviours of a participant and the relationships express the peer-to-peer links between participants.
- `<exchange>`: this tag defines the variables exchanged by the sender and the receiver. It contains two elements `<send>` and `<receive>` where the former defines the information sent by the sender whereas the latter defines the information received by the receiver. The attribute `action` defines the direction of the interaction and has two possible values: `request` or `respond`. In the first case the interaction is performed from the role `fromRole`, which is the sender, to the role `toRole` which is the receiver. In the second case the `toRole` is the sender and the `fromRole` is the receiver.
  One or two exchange tags can be defined within the tag interaction. If there are two exchange elements one must have action equal to request and the other must assume the value respond. The former defines the information exchange during the request interaction and the latter during the response one.

The description of the main attributes of the `<interaction>` tag follows:

- The attributes `channelVariable` and `operation` define the WS-CDL channel and the WSDL operation on which the interaction is performed and which has to belong to the interaction target.
- The attribute `align` defines if the interaction must be aligned or not. In the following we quote from the specification[7] the definition of the align attribute:

> If the align attribute is set to "false" for the Interaction, then it means that the:
> - Request exchange completes successfully for the requesting Role once it has successfully sent the information of the Variable specified within the send element and the Request exchange completes successfully for the accepting Role once it has successfully received the information of the Variable specified within the receive element
> - Response exchange completes successfully for the accepting Role once it has successfully sent the information of the Variable specified within the send element and the Response exchange completes successfully for the requesting Role once it has successfully received the information of the Variable specified within the receive element

> If the align attribute is set to "true" for the Interaction, then it means that the Interaction completes successfully if its Request and Response exchanges complete successfully[...]:
> - A Request exchange completes successfully once both the requesting Role has successfully sent the information of the Variable specified within the send element and the accepting Role has successfully received the information of the Variable specified within the receive element

- A Response exchange completes successfully once both the accepting Role has successfully sent the information of the Variable specified within the send element and the requesting Role has successfully received the information of the Variable specified within the receive element

This definition is far to be formal and deserves to be commented. An interaction is aligned when both the roles are aware of its state. On the contrary, it must be considered not aligned when the sender and the receiver act without a common knowledge about the interaction state. From the choreography point of view, that is a system top view, a common knowlegde about the interaction state is linked to the fact that it is possible to verify its termination or not. For this reason here we discriminate between aligned interaction and the not aligned one considering the fact that it is possible to control their termination or not.

Considering the meaning of the WS-CDL `<interaction>` tag, we extract three kinds of interaction patterns:

- *request*
- *splitted request-respond*
- *atomic request-respond*

The three type of interactions patterns *request*, *splitted request-respond* and *atomic request-respond* directly follow from the `<exchange>` elements within the `<interaction>` tag.

The *request* pattern is characterized by an interaction defined with only one exchange element and the action set to request. The operation through which the interaction is performed must be a One-Way operation.

In the following we present the WS-CDL code for a request interaction pattern where `opAB` is a One-Way operation:

```
<interaction name="interactionAB" channelVariable="tns:ABchan"
      operation="tns:opAB" align="true" >
      <participate relationshipType="tns:relationshipAB"
       fromRole="tns:roleA" toRole="tns:roleB"/>
      <exchange name="requestAB" action="request">
          <send variable="cdl:getVariable("tns:x", "", "")" />
          <receive variable="cdl:getVariable("tns:y", "", "")"/>
      </exchange>
</interaction>
```

The *splitted request-respond* pattern is characterized by two different interactions defined on the same Request-Response operation: one for expressing the request interaction and the other one for expressing the respond one. This pattern allows to program the request-response interaction in a low level manner where the two single interactions can be managed separately. In the following we present the WS-CDL code for a splitted request-respond pattern where a choreography `C` is performed between the request and the respond interactions:

```
<sequence>
   <interaction name="interactionCD"  channelVariable="tns:CDchan"
      operation="tns:opCD" align="true" >
```

```
      <participate relationshipType="tns:relationshipCD"
                fromRole="tns:roleC" toRole="tns:roleD"/>
      <exchange name="requestCD" action="request">
            <send variable="cdl:getVariable("tns:z", "", "")" />
            <receive variable="cdl:getVariable("tns:k", "", "")"/>
      </exchange>
   </interaction>
   <perform>
   <!--  Choreography C -->
   </perform>
   <interaction name="interactionCD2"  channelVariable="tns:CDchan"
      operation="tns:opCD" align="true" >
      <participate relationshipType="tns:relationshipCD"
            fromRole="tns:roleC" toRole="tns:roleD"/>
      <exchange name="responseCD" action="respond">
          <send variable="cdl:getVariable("tns:w", "", "")" />
          <receive variable="cdl:getVariable("tns:q", "", "")"/>
      </exchange>
   </interaction>
</sequence>
```

The *atomic request-respond* pattern is characterized by two exchange elements defined within the interaction tag. One represents the request exchange and the other represents the respond. The operation through which the interaction is performed must be a Request-Response one. In the following we present the WS-CDL code for an atomic request-respond pattern where `opCD` is a Request-Response operation:

```
<interaction name="interactionCD"  channelVariable="tns:CDchan"
      operation="tns:opCD" align="true" >
      <participate relationshipType="tns:relationshipCD"
             fromRole="tns:roleC" toRole="tns:roleD"/>
      <exchange name="requestCD" action="request">
        <send variable="cdl:getVariable("tns:z", "", "")" />
        <receive variable="cdl:getVariable("tns:k", "", "")"/>
      </exchange>
      <exchange name="responseCD" action="respond">
        <send variable="cdl:getVariable("tns:w", "", "")" />
        <receive variable="cdl:getVariable("tns:q", "", "")"/>
      </exchange>
</interaction>
```

Moreover, each interaction can be enriched with the alignment property setting the align attribute of the tag interaction. In the examples above we have considered all aligned interactions.

Now we are ready to present the syntax of $WSCDL_{core}$ which deals with such a kind of patterns. We start by modeling only the request and splitted request-respond patterns without considering the atomic request-respond one. Such a kind of pattern will be separately discussed in section 3 where we show that it is a special case of the splitted request-respond pattern.

Let us now present the syntax of $WSCDL_{core}$:

$$C_{core} ::= \mathbf{0} \mid \mu \mid \mu^A \mid C_{core}; C_{core} \mid C_{core} \parallel C_{core} \mid C_{core} + C_{core}$$

$$\mu ::= \texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}) \mid \texttt{respond}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$$

A conversation can be a terminated conversation **0**, a not aligned interaction $\mu$, an aligned interaction [2] $\mu^A$ or the sequential, parallel and alternative composition of conversations.

The interactions can be a `request` interaction or a `respond` one. Considering their arguments we have that:

- $\rho_A$ and $\rho_B$ represent the invoker role and the invoked one, respectively.
- $o$ is the operation through which the interaction is performed and which has to be exhibited by the invoked role $\rho_B$.
- $\widetilde{x}$ and $\widetilde{y}$ are respectively the variables of $\rho_A$ and $\rho_B$.

When a request interaction completes, the direction of the interaction is from role $\rho_A$ to role $\rho_B$ and $\widetilde{x}$ will populate[3] $\widetilde{y}$ whereas when a respond interaction completes the direction is from role $\rho_B$ to role $\rho_A$ and $\widetilde{y}$ will populate $\widetilde{x}$. Depending on the type of the operation $o$ the respond interaction is allowed or not. If $o$ is a One-Way operation only a request interaction is allowed on that operation whereas if $o$ is a Request-Response operation the two interactions are allowed and the respond one must logically follow the request. When an interaction is not aligned it is not possible to control when it completes whereas for the aligned one this is possible.

Finally, conversations can be: i) the sequential composition of two conversations $C_{core}; C'_{core}$ whose meaning is that $C'_{core}$ can be performed after $C_{core}$ completes, ii) the parallel composition of two conversations $C_{core} \parallel C'_{core}$ which represents the concurrent execution of conversations $C_{core}$ and $C'_{core}$, and iii) the alternative composition of two conversations $C_{core} + C'_{core}$ whose meaning is that the conversation to be performed is non-deterministically selected between $C_{core}$ and $C'_{core}$.

Now we define a choreography. A choreography, denoted by $CH_{CDL}$, is defined by a pair $(C_{core}, \Sigma)$ where $C_{core} \in WSCDL_{core}$ and $\Sigma \subseteq Role$. Here we consider only well-formed choreographies whose definition follows:

**Definition 1 (Well-formed set of roles).** *Let $\Sigma \subseteq Role$. The set of roles $\Sigma$ is well-formed if the following conditions hold:*

1. *$\Sigma$ is finite;*
2. *if $(\rho_i, \omega_i, \sigma_i) \in \Sigma$ and $(\rho_j, \omega_j, \sigma_j) \in \Sigma$ and $\rho_i = \rho_j$ then $i = j$.*

**Definition 2 (Well-formed choreography).** *Let $(C_{core}, \Sigma)$ be a choreography; $C_{core}$ is well-formed if:*

[2] Even if the correct notation is $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})^A$ and $\texttt{respond}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})^A$, for the sake of clarity, we will use the notation $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$ and $\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$.

[3] In WS-CDL slang the term populate means that the values instantiated by the variables $\widetilde{x}$ will be stored within the variables $\widetilde{y}$. Since we use variables as names, in the following we will exploit the term populate in order to express the fact that an interaction represents an information flow from variables $\widetilde{x}$ to variables $\widetilde{y}$

1. *for any operation* $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$ *and* $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$ *it contains, the following conditions hold:*
   *(a)* $(\rho_A, \omega_A, V_A), (\rho_B, \omega_B, V_B) \in \Sigma$ *for some* $\omega_A, V_A$ *and* $\omega_B, V_B$*;*
   *(b)* $(o, ow) \in \omega_B \vee (o, rr) \in \omega_B$*;*
   *(c)* $\widetilde{x}$ *and* $\widetilde{y}$ *have the same arity;*
   *(d)* $\widetilde{x} \subseteq V_A$ *and* $\widetilde{y} \subseteq V_B$
2. *for any operation* $\texttt{respond}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$ *and* $\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})$*:*
   *(a)* $(\rho_A, \omega_A, V_A), (\rho_B, \omega_B, V_B) \in \Sigma$ *for some* $\omega_A, V_A$ *and* $\omega_B, V_B$*;*
   *(b)* $(o, rr) \in \omega_B$*;*
   *(c)* $\widetilde{x}$ *and* $\widetilde{y}$ *have the same arity;*
   *(d)* $\widetilde{x} \subseteq V_A$*,* $\widetilde{y} \subseteq V_B$

Definition 1 states that a set of roles $\Sigma$ is well-formed if it is finite and the set of role names are all distinct.

Definition 2 states constraints for the interactions. Conditions *(a)* require that the roles involved are contained in the system, conditions *(b)* guarantee that the role which receives the interaction exhibits the operation used to interact (in the case of a respond interaction the type of the operation must be Request-Response), conditions *(c)* ensure that the sender and the receiver use the same number of variables and, finally, conditions *(d)* ensure that the specified variables belong to the corresponding role.

### 2.3 The semantics

The semantics of $WSCDL_{core}$ is presented in terms of the semantics of an auxiliary language $CL_P$ which has been already used to define the semantics of our conversation language $CL$.

**Syntax of $CL_P$** The auxiliary language is defined by the following grammar:

$$C_P ::= \mathbf{0} \mid \mu \mid C_P; C_P \mid C_P \parallel C_P \mid C_P + C_P$$
$$\mu ::= (\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, dir)$$

In the following we use $CL_P$, ranged over by $C_P$, to denote the set of conversations of such a language. We limit the description to the interaction $\mu$ since the composition operators ;,+,$\parallel$ have the same meaning of those of $WSCDL_{core}$.

$(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, dir)$ means that an interaction from role $\rho_A$ to role $\rho_B$ is performed. In particular, $o$ is the name of the operation $(o, t) \in Operation$ on which the message exchange is performed. Variables $\widetilde{x}$ and $\widetilde{y}$ are those used by the sender and the receiver, respectively. When the interaction completes, it is assumed that the information represented by the variables $\widetilde{x}$ will populate the variables $\widetilde{y}$. Finally, $dir \in \{\uparrow, \downarrow\}$ indicates whether the interaction is a request ($\uparrow$) or a response ($\downarrow$) of $o$. Thus the $dir$ parameter is needed for allowing us to reason on simple interaction and at the same time for preserving information about the type of the operation on which the message exchange is performed.

**Semantics of $CL_P$** $C_P \rightarrow C'_P$ means that the conversation $C_P$ evolves in one step in a conversation $C'_P$. We define $\rightarrow$ as the least relation which satisfies the axioms and rules of Table 1 and closed w.r.t. $\equiv$.

$$\frac{}{(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, dir) \rightarrow \mathbf{0}}\ (\textsc{Interaction}) \qquad \frac{C_P \rightarrow C'_P}{C_P; D_P \rightarrow C'_P; D_P}\ (\textsc{Sequence})$$

$$\frac{C_P \rightarrow C'_P}{C_P \mid D_P \rightarrow C'_P \mid D_P}\ (\textsc{Parallel}) \qquad \frac{C_P \rightarrow C'_P}{C_P + D_P \rightarrow C'_P}\ (\textsc{Choice})$$

(Structural Congruenge)

$$\mathbf{0}; C_P \equiv C_P \qquad C_P \mid \mathbf{0} \equiv C_P \qquad C_P + \mathbf{0} = C_P$$
$$C_P + D_P \equiv D_P + C_P \quad C_P \mid D_P \equiv D_P \mid C_P$$
$$(C_P + D_P) + E_P \equiv C_P + (D_P + E_P)$$
$$(C_P \mid D_P) \mid E_P \equiv C_P \mid (D_P \mid E_P)$$

**Table 1.** Semantics of $CL_P$ conversations

The structural congruence $\equiv$ is the least congruence closed w.r.t. the axioms of Table 1 which express the fact that abelian monoid laws for parallel and choice operators hold (associativity, commutativity and $\mathbf{0}$ as identity) and the sequence operator $C_P; D_P$ enables $D_P$ only when $C_P$ completes.

The description of axioms and rules follows. The axiom Interaction describes the behavior of an interaction. $\rho_A$ is the name of the sender role whereas $\rho_B$ is the name of the receiver one, while $o$ is the operation name used to interact and $\widetilde{x}$, $\widetilde{y}$ are the variables used by the sender and the receiver to exchange data, respectively. The rules Sequence, Parallel and Choice are standard.

**Mapping of $WSCDL_{core}$** We define a mapping from $WSCDL_{core}$ conversations into terms of $CL_P$. Definition 3 defines the mapping function. Request and respond interactions are mapped in an interaction of $CL_P$ where in the first case the direction parameter is $\uparrow$ and in the second is $\downarrow$. The rule 4 stands that either the aligned and the not aligned interactions are implemented in the same way. Their different behaviour is strictly related to the fact that it is not possible to control when the not aligned one completes. Such a difference is expressed in rules 5 and 6 where the sequence is preserved only in the case of the aligned interaction. In the opposite case, the sequence operator is replaced by the parallel one. Indeed, the sequence of a not aligned interaction with another conversation is implicitly a parallel composition of them because, since it is not possible to control when the not aligned interaction completes, it is impossible to express

the fact that a certain conversation have to be performed after its completion. Thus, it is impossible to support sequencing of not aligned interaction.

For example, let $C = \texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); \texttt{request}^A(\rho_C, \rho_D, o', \widetilde{z}, \widetilde{k})$ be a conversation. The fact that the first `request` is not aligned implies that it is not possible to control when it completes. On the other hand, the sequence operator guarantees that the conversation which follows can be executed only when the previous one has completed. However, in this case such a condition cannot be tested. This means that it is not possible to control the sequential execution of not aligned interactions, therefore we map not aligned interactions composed in sequence by replacing the operator with the parallel one. Consequently, exploiting the rule 6 of the mapping function, the conversation $C$ behaves as:

$$[\![C]\!] = [\![\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})]\!] \parallel [\![\texttt{request}^A(\rho_C, \rho_D, o', \widetilde{z}, \widetilde{k})]\!]$$

Finally the mapping preserves the parallel and the alternative composition.

**Definition 3 (The mapping function).** *The function* $[\![\ ]\!] : WSCDL_{core} \rightarrow CL_P$ *is defined inductively as follows:*

1. $[\![\boldsymbol{0}]\!] = \boldsymbol{0}$
2. $[\![\boldsymbol{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})]\!] = (\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \uparrow)$
3. $[\![\boldsymbol{respond}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})]\!] = (\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \downarrow)$
4. $[\![\mu]\!] = [\![\mu^A]\!]$
5. $[\![\mu^A; C_{core}]\!] = [\![\mu^A]\!]; [\![C_{core}]\!]$
6. $[\![\mu; C_{core}]\!] = [\![\mu]\!] \parallel [\![C_{core}]\!]$
7. $[\![C_{core} \parallel C'_{core}]\!] = [\![C_{core}]\!] \parallel [\![C'_{core}]\!]$
8. $[\![C_{core} + C'_{core}]\!] = [\![C_{core}]\!] + [\![C'_{core}]\!]$

## 3 The request-respond patterns

This section is devoted to reason about the two request-respond interaction patterns supported by WS-CDL and represents the main contribute of this paper. In the first part we criticize the splitted request-respond pattern because it is not reasonable to specify separately the alignment property for the request and the respond. In the second part we model the atomic request-respond pattern where the alignment property is referred to the entire interaction showing that this is the right granularity level where alignment property has to be defined. Furthermore, we show that the atomic request-respond interaction has some lacks of expressiveness w.r.t. the splitted one.

### 3.1 Splitted request-respond pattern

As confirmed by the choreography working group in the official mailing list [8], *a splitted request-respond pattern programmed in a conversation, say $C$, is considered* valid *when, for all the possible computation paths of $C$, the request is executed before the respond, and in the case the request is performed the corresponding respond is executed.* This directly follows by the fact that it is strictly related with the Request-Response operations where the response is subordinate to the request. Thus we can express splitted request-respond interactions in the following ways:

a) $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$
b) $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$
c) $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$
d) $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$

where $C[\,]$ is any conversation context. Here we intend to reason on the meaning of the alignment property for a splitted request-respond pattern, referring in particular on cases a) and b) where the request is not aligned. The following proposition expresses that any interaction with a not aligned request on a Request-Response operation is not valid. The idea directly follows from the semantics of the not aligned interaction. Indeed, in the cases a) and b) we cannot guarantee that a not aligned request is performed before the corresponding respond. Consequently, in a request-respond interaction we always have to use aligned requests.

**Proposition 1.** *Let $(o, rr) \in Op$ be an operation. For any conversation context $C[\,]$, for any $\rho_A$ and $\rho_B$, for any $\widetilde{x}$ and $\widetilde{y}$, the conversations*

- $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$
- $\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C[\texttt{respond}(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]$

*are not valid.*

Let us consider the following example where $C$ is the following conversation in order to clarify the meaning of the notion above:

$$C = \texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); \texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})$$

$C$ behaves as follows:

$$[\![C]\!] = [\![\texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})]\!] \parallel [\![\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]\!]$$

It is easy to observe that such an interaction is not valid because it is trivial to prove that there exist a computation path where the respond is performed before the request.

Thus, in a splitted request-respond interaction the request must be always aligned. Now we can assume that the aligned property shifts always on the respond interaction. The cases c) and d) above express this concept.

The case c) represents the aligned interaction whereas the case d) the not aligned one. Let us now consider the case d), does it make sense? Although it is syntactically correct, in our opinion, it represents a wrong interpretation of the request-respond interaction. The alignment property indeed means that we cannot control when an interaction completes. In the case d) we can control that the first part of a request-respond interaction completes leaving the not aligned behaviour only on the second one. Here we cannot state if WS-CDL authors were willing to allow such a behaviour but, in our opinion, the case d) does not address the nature of the alignment because we interpret a request-respond interaction, even splitted, as a unique concept where a property must hold on the entire interaction.

We conclude that, in our opinion, it is not reasonable to express alignment property at the level of single message exchange. For these reasons the splitted request-respond pattern is inadequate for expressing the alignment property.

### 3.2 Atomic request-respond pattern

In this section we introduce the atomic request-respond pattern showing that it defines only valid conversations and allows to express the alignment property at the granularity level of the entire request-respond interaction which seems to be the right approach.

Here we extend the core language of Section 2.2 with the atomic request-respond pattern where the interactions $\mu$ are defined as it follows:

$$\mu ::= \texttt{request}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}) \mid \texttt{respond}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}) \mid \texttt{rr}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k})$$

The atomic request-respond, denoted with $\texttt{rr}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k})$, means that $\rho_A$ performs a request-respond with $\rho_B$ by using variables $\widetilde{x}$ (of $\rho_A$) and $\widetilde{y}$ (of $\rho_B$) for the request message and variables $\widetilde{z}$ (of $\rho_A$) and $\widetilde{k}$ (of $\rho_B$) for the response.

The semantics of the extended language is obtained by adding the following encoding rule to the ones of Definition 3:

$$\begin{aligned}&[\![\texttt{rr}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k})]\!] = \\ &\quad [\![\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y})]\!]; [\![\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})]\!]\end{aligned}$$

The atomic request-respond pattern is expressed in terms of the splitted one; it is trivial to verify that such kind of interactions are all valid accordinlgy with the notion given in section 3.1. Indeed, the request is aligned as well as the respond thus the sequential operator guarantees that the request is always performed before the respond.

So far, we have only presented the semantics of $\texttt{rr}^A$ without discussing that of `rr`. This is due to the fact that it is implicitly linked to the semantics of the alignment property. What does it mean performing a not aligned atomic request-respond? In this case we cannot control when the entire request-respond interaction completes. The semantics of such a condition follows from rules 4, 5 and 6 of the mapping function:

$$[\![\texttt{rr}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k}); C]\!] = ([\![\texttt{rr}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k})]\!] \parallel [\![C]\!]$$
$$[\![\texttt{rr}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k}); C]\!] = [\![\texttt{rr}(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}, \widetilde{z}, \widetilde{k})]\!]; [\![C]\!]$$

Here the alignment property is expressed at the level of the entire request-respond interaction contrarily to the case of the splitted one. Considering the two request-respond patterns, we conclude that the right granularity level for expressing the alignment property is represented by the atomic request-respond pattern.

Let us now consider the expressiveness of this construct w.r.t. the splitted one. Considering the semantics of the atomic request-respond pattern it is trivial to observe that it is less expressive than the splitted one. The following examples prove that there exist an expressiveness gap because they cannot be programmed by using the atomic pattern.

1. $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C; \texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})$
2. $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); (\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k}) \parallel D)$
3. $\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C; (\texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k}) \parallel D)$

where $C$ is a choreography which is performed between the request and the respond interactions and $D$ is a choreography which is executed in parallel with the respond interaction and after the request interaction accordingly with the semantics.

Summarizing, the atomic request-respond interaction pattern on the one hand has the right granularity level for expressing the alignment property and on the other hand it lacks expressiveness.

## 4 Conclusions

In this paper we have presented the formal semantics of a significant fragment of WS-CDL which provides a mean to deal with interactions. We have analysed the adequacy of such interaction patterns when the alignment property is considered by showing that the request-respond patterns do not represent the best choice. Indeed, the splitted one permits to align the single message exchange which, on the one hand allows to express not valid conversations and, on the other hand implements the not aligned property in a way which, in our opinion, does not address the nature of the alignment property. The atomic request-respond pattern on the contrary, provides the right level to reason about alignment but presents significant lacks of expressiveness (e.g. it is not possible to express that a certain conversation must be performed during the execution of a request-respond).

To the best of our knowledge, this is the first attempt towards the definition of interaction patterns in choreography when aligned property is considered. The only works which deal with choreography languages are [2] and [7]. In the former Web Service Choreography Interface (WSCI) [10] language is modeled whereas in the latter our formal language $CL$ is presented.

$CL$, whose semantics is expressed in terms of $CL_P$, supports two interaction patterns where one is exactly the request of $WSCDL_{core}$ while the other one, used to program the request-respond, has a semantics which sounds like:

$$\texttt{request}^A(\rho_A, \rho_B, o, \widetilde{x}, \widetilde{y}); C; \texttt{respond}^A(\rho_A, \rho_B, o, \widetilde{z}, \widetilde{k})$$

such a pattern has the same granularity level of the atomic one of $WSCDL_{core}$ and covers some of its lacks of expressiveness. Indeed, it is trivial to verify that with this pattern it is possible to express an inner conversation between the request and the respond (example 1 of section 3.2). In light of these observations $CL$ can be considered a good starting point for representing choreography.

As future works we intend to reason on other aspects of WS-CDL and at the same time to improve our formal language $CL$. Furthermore, we are interested to investigate the relationships which exist between the choreography approach and the orchestration one by introducing a notion of conformance between the two models starting by studying how the choreography alignment property affects a related orchestration.

## References


1. Mario Bravetti, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Supporting e-commerce systems formalization with choreography languages. In *SAC*, pages 831–835, 2005.
2. A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web services choreographies. In M. Bravetti and G. Zavattaro, editors, *Proc. of 1st International Workshop on Web Services and Formal Methods (WS-FM 2004)*, volume 105 of *ENTCS*. Elsevier, 2004.
3. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Towards a formal framework for Choreography. In *Proc. of 3rd International Workshop on Distributed and Mobile Collaboration (DMC 2005)*. IEEE Computer Society Press. To appear.
4. F. Leymann. Web Services Flow Language (WSFL 1.0). [http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf], Member IBM Academy of Technology, IBM Software Group, 2001.
5. Microsoft,IBM, Siebel Systems, BEA. *Business Process Execution Language for Web Services Version 1.1.* [http://www-106.ibm.com/developerworks/library/ws-bpel/].
6. S. Thatte. XLANG: Web Services for Business Process Design. [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm], Microsoft Corporation, 2001.
7. W3C. *Web Services Choreography Description Language Version 1.0. Working draft 17 December 2004.* [http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/].
8. W3C. *Web Services Choreography Working Group, public mailing list.* public-ws-chor@w3.org.
9. W3C. *Web Services Description Language (WSDL) 1.1.* [http://www.w3.org/TR/wsdl].
10. World Wide Web Consortium (W3C). Web service choreography interface (wsci) 1.0. [http://www.w3.org/TR/wsci], 2002.