Viewpoints for Modeling Choreographies in Service-Oriented Architectures

Sebastian Wieczorek[1], Andreas Roth[1], Alin Stefanescu[1], Vitaly Kozyura[1],
Anis Charfi[1], Frank Michael Kraft[2], Ina Schieferdecker[3]

[1] SAP Research,
Bleichstr. 8,
64283 Darmstadt, Germany
firstname.lastname@sap.com

[2] SAP AG,
Dietmar-Hopp-Allee 16,
69190 Walldorf, Germany
frank.michael.kraft@sap.com

[3] Fraunhofer FOKUS,
Kaiserin-Augusta-Allee 31,
10589 Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

Abstract

Component integration plays a decisive role in service-oriented architectures (SOAs). The technical implementation must faithfully reflect business and enterprise integration requirements. This implies a good understanding of the globally observable message choreography but also of how messages are handled by the involved components and by the SOA middleware. In this paper we present a solution to the problem of keeping global and local viewpoints in synchronization via a common message choreography metamodel. As main contribution we shape various interpretations of global choreographies, which were left unspecified in state-of-the-art choreography approaches. We have implemented a message choreography modeling (MCM) environment incorporating these contributions. MCM seamlessly complements existing models at SAP. We show how service integration experts, architects, and testers can benefit from our approach that enables model-based integration testing and model verification facilities.

1. Introduction

Enterprise Resource Planning (ERP) software [10] integrates different organizational parts and functions into one logical software system, with SAP being a leading provider of ERP software. Service-oriented architectures (SOA) are recently regarded as the next evolutionary step to cope with the ever increasing complexity of ERP systems.

SOA provides methods and frameworks to compose single services in order to realize complex business scenarios. Modeling and implementation of such services based on technical specifications like XML, SOAP, and WSDL is well-understood. The challenging part of SOA implementations is the integration of different services according to the defined business processes. At the lower end, one service is described as a set of operations and message types, its functioning relying on a simple request-response pattern. At the service integration level, more complicated specifications are needed to capture not only the message exchanges and their underlying message types but also the dependencies between these exchanged messages, i.e., both control-flow and data-flow dependencies. Choreography languages like WS-CDL [8], BPMN [1], or Let's Dance [16] were introduced to describe the interaction protocols between a set of loosely coupled components communicating over message channels from the perspective of a global observer.

There are three different orthogonal viewpoints that have to be considered to fully understand these service interactions for a smooth integration. These are described as follows:

Static vs. dynamic viewpoints. This differentiation is a well understood aspect: *static* (or *structural*) specifications of a service-oriented architecture provide descriptions of the components involved in the collaboration. Their interface definitions include message types and the available communication channels. The *dynamic* (or *behavioral*) specifications describe the order in which the messages could or should be exchanged. Dynamic specifications can be written in dedicated choreography languages (like WS-CDL [8]), orchestration languages (like BPEL [2]) or general purpose ones (like BPMN [1] and UML behavioral diagrams).

Message exchange viewpoints. Regarding the message exchange, it is important to precisely specify what kind of events is described in a behavioral specification: the message send, receive, or observation events. Usually this aspect is neglected in the literature where most approaches implicitly

assume a send semantics. However, such a semantics does not reflect possible changes of the message order when exchanging messages over unreliable channels, while explicit information would support service integration testing much better. Therefore we investigate the different capabilities of message exchange viewpoints in Section 4.

Global vs. local viewpoints. This aspect received a lot of attention in literature. Service interactions can be seen from a global perspective or from the local perspectives of the involved components. The former is usually described using choreographies, whereas the latter using orchestrations. Keeping all these perspectives consistent is a major challenge of choreography modeling. In this paper we provide some insights into the considerations of consistency enforcement for these perspectives from an enterprise architecture and development lifecycle point of view. Our approach is discussed in Section 5 and Section 6.

The contributions of this paper are:

- An investigation of three different viewpoints for the semantics of message exchange based on send, observe, or receive interpretations and in which contexts they are useful.
- A technical solution based on a common metamodel that bridges the local and global viewpoints of the choreography.
- An exemplification of the above concepts on a new domain-specific language from SAP, Message Choreography Models (MCM). This language is based on the requirements we defined in previous work [13] (i.e., graphical state-based representation, explicit concurrency, detailed message types, local viewpoints, determinism, a distinction between global and local constraints). This was necessary because we observed that state of the art choreography languages do not consider all these requirements simultaneously.
- A short description of how the introduced methods are motivated by the model-based testing and verification requirements.

The remainder of this paper is structured as follows. Section 2 introduces a running example used throughout the paper to illustrate our approach. Section 3 presents the MCM language. Section 4 investigates message exchange viewpoints. In Section 5, a solution for global and local viewpoints is proposed, while Section 6 focuses on considerations when using the approach in the development process. Section 7 reports on related work. Section 8 concludes the paper and gives future work directions.

2. A Buyer-Seller Example at SAP

This section introduces a running example from the enterprise world, describing a simplified communication between a buyer and a seller. In the next sections we will refer to it in order to explain the different viewpoints and how we address them.

For better understandability we first sketch SAP's architecture and modeling framework. SAP's approach to SOA [15] can be exemplified by the SAP solution "Business ByDesign"[1], which was fully developed following SOA principles and model-driven methodologies [7]. The approach uses metamodels for describing the SOA artifacts, a governance process and a modeling tool environment [7]. The main architectural building blocks in the metamodel are business objects (BO) and process components (PC). The BOs provide the core business logic, several related BOs are grouped into PCs. The BOs provide their functionality via core services, which are bundled at the PC level into composed services. Process Component Interaction Models (PCIM) describe the message types exchanged between the PCs and their associated (composed) service interfaces for exactly two communicating PCs. Integration Scenario models provide a bird-eye view of all PCs participating in business scenarios and how they are connected.

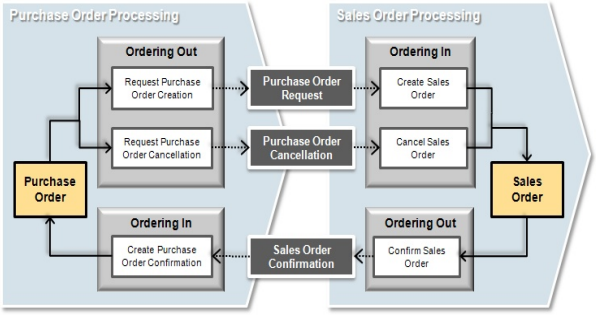


Figure 1. PCIM for the buyer-seller example

A simplified PCIM, provided in Figure 1**,** has two PCs: *Purchase Order Processing* and *Sales Order Processing*. The buyer controls the Purchase Order component while the Sales Order component is associated with the seller. Each PC contains one BO: the *Purchase Order* and *Sales Order*, respectively. The information exchange is realized by the service interfaces *Ordering Out* and *Ordering In* using three message types for request, cancellation and confirmation. A PCIM provides only a static viewpoint of the service interaction, and so an extra

[1] *http://www.sap.com/solutions/sme/businessbydesign*

model is needed to describe the dynamic viewpoint, the message choreography between two instances of the BOs.

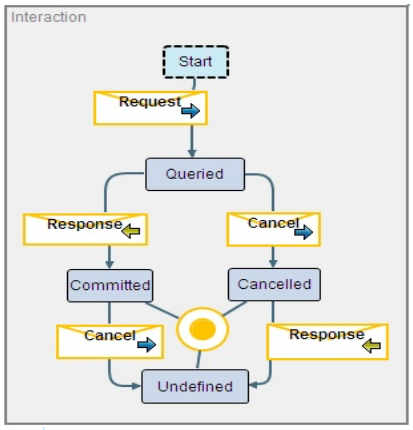


Figure 2. A simple buyer-seller choreography

A choreography model (in the MCM language described in Section 3) for this PCIM is shown in Figure 2. The buyer requests a certain product using a *request* message (*Purchase Order Request*). Consequently, either the seller accepts the order by sending the *response* message (*Sales Order Confirmation*) or the buyer stops the transaction by sending a *cancel* message (*Purchase Order Cancellation*).

The rounded rectangles represent the current state of the choreography while the envelopes stand for the messages being exchanged. The arrows on the envelopes define the sender of the message: left-to-right for the buyer and right-to-left for the seller. The circle in the center marks the valid target states of the conversation. The precise semantics of the models is given in next section.

As pointed out in [13], in our context it is beneficial to restrict choreographies to exactly two partners. This is also consistent with the existing PCIMs of SAP architecture (see Figure 1) where exactly two communication partners are involved. An extension of MCM to more than two partners is feasible and is subject of future work.

Even though the presented protocol description is precise enough for a high-level business view of the process, some semantical subtleties have to be considered. For example it remains unclear whether this description specifies a subset of the intended behavior, which may allow additional transitions (e.g. observing a new *request* message in the state *committed*), or a maximal allowed behavior, such that conformant implementations are allowed to leave out some functionality like not sending a *response* after a *cancel*? Moreover, the semantics of message sending and receiving has to be clearly defined based on the specific channel assumptions (see Sect. 3.3).

3. Message Choreography Models (MCM)

The Message Choreography Modeling language MCM developed at SAP Research is described in this section. It will serve to illustrate the different viewpoints motivated before. MCM is a domain-specific language which smoothly integrates into a set of other proprietary modeling languages adapted to the architectural settings at SAP [7]. It adheres to specific requirements for test case generation and verification which are not fulfilled by existing choreography languages [13]. We implemented an Eclipse-based editor for MCM (all figures in this paper are screenshots) and plug-ins for verification and model-based testing. The editor is integrated with SAP tooling and allows for importing existing SAP models like PCIMs or BOs.

MCM consists of the following three types of models:

- *global choreography models* specifying a high-level view on the exchanged messages,
- *local partner models* describing separately the behavior of each of the involved parties, and
- *channel models* stipulating characteristics of the communication channel on which messages are exchanged between the partners.

After defining these models, we will discuss their relationships and different viewpoints based on them.

3.1 Global Choreography Model

Syntax. Following the original idea of choreography models as a supra-component view on message exchange our *global choreography models* (GCM) provide a high level view on service interaction. One does not describe single “sending” or “receiving” messages events, but takes the position of a virtual observer between the components. This observer monitors the exchanged messages and is agnostic of the activities inside the involved components.

The GCM corresponds to a labeled transition system (LTS) where the transitions are labeled by *interactions*, i.e. events of observing a message on the

channel. The states are *global states* of the communication specifying whether a certain message exchange could be observed at that stage. To simplify the presentation, the global state space for now just consists of a finite set of named states with one distinguished initial state. Note that in our implementation we have a more sophisticated state space (similar to an extended concurrent finite state machine) described in Sect. 3.5.

Interactions consist of four ingredients:

- A message type identified by a unique name.
- The sender and the receiver of the interaction.
- An effect on the global state, i.e. the subsequent global state which is activated when the interaction takes place.
- A guard condition restricting when the interaction is enabled. For now, we assume that the condition only defines in which state the interaction can take place. In Sect. 3.5 more detailed guard conditions are presented.

Target states are global states that describe when the modeled choreography can be terminated, i.e. when it is legal that no further interaction takes place. In some real life scenarios it cannot be decided whether a service conversation reached an end state, especially if one of the participants is able to restart the conversation. Our target states therefore differ from end states of existing choreography languages, where termination means that no further interaction can take place, but they are similar to the classical notion of accepting/final states from finite automata theory.

A GCM for a set of message types consists of a set of partners designated as senders or receivers of these message types, a set of states, some of which are marked as initial or as target states, and a set of interactions. Figure 2 gives an example of a GCM. The dark boxes represent the states; the one with a dashed border is initial. Interactions are depicted as envelopes associated with an arrow between states and labeled with the message type of the interaction and the direction of message exchange. The target of the arrow associated with an interaction represents the effects of the interaction. For instance, *Request* has the effect of reaching the state *Queried*. The source of the arrow indicates that the state represents the guard of the interaction. The target states are connected to the filled circle (e.g. the target states in Figure 2 are *Committed*, *Cancelled* and *Undefined*).

Semantics. Given a GCM *G*, we denote by *LTS(G)* the labeled transition system of *G*. The *traces* of *G*, denoted by *Traces(G)*, is the set of sequences $(i_1,\ldots,i_n)$ of interactions for which there exists a sequence $(s_0,\ldots,s_n)$ of states such that s_0 is initial, for all $k=1,..,n$, s_{k-1} satisfies the guard of i_k and s_k is the effect of i_k , and s_n is a target state. In Sect. 4, we define more precisely the relation between the atomic interactions of *G* and their corresponding sending and receiving events.

3.2 Local Partner Model

A *local partner model* (LPM) focuses on a single component involved in a choreography, describing which messages may be *sent* and *received* by that communication partner. Very similar to GCMs, the main elements of an LPM are a set of *send events*, and a set of *receive events* as well as a set of named *local states* with a distinguished initial state. Local target states are a subset of the local states. Similar to the interaction in GCM a *send event* (respectively a *receive event*) consists of the message type, the receiving partner (respectively the sending partner), effect and guard defined on the local state space.

Figure 3 gives an example of two LPMs, one for the buyer and one for the seller. The send and receive events are distinguished graphically by upwards and downwards pointing arrows, respectively.

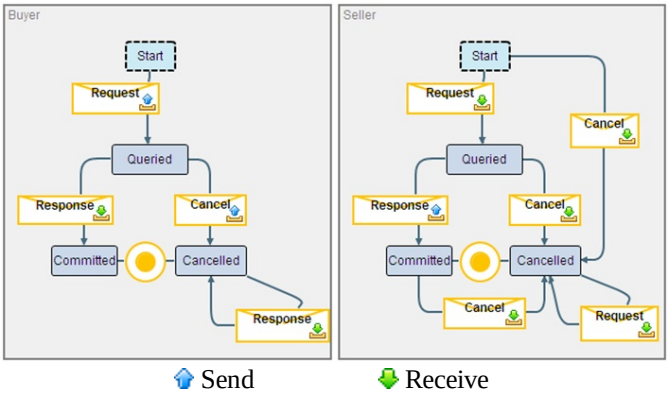


Figure 3. Example of LPMs for Buyer and Seller

3.3 Channel Model

Given a set of message types *MT*, a *channel model* is a total function *r* from a sequence of messages (of *MT*) to a sequence of messages (of *MT*). Given *MT'* a subset of *MT* and s a message sequence, $\pi_{MT'}(s)$ denotes the projection of *s* to sequences of the messages of message types *MT'*. Let $\pi_{MT'}$ be canonically extended on the channel model. The channel model *r* is then based on assignments of

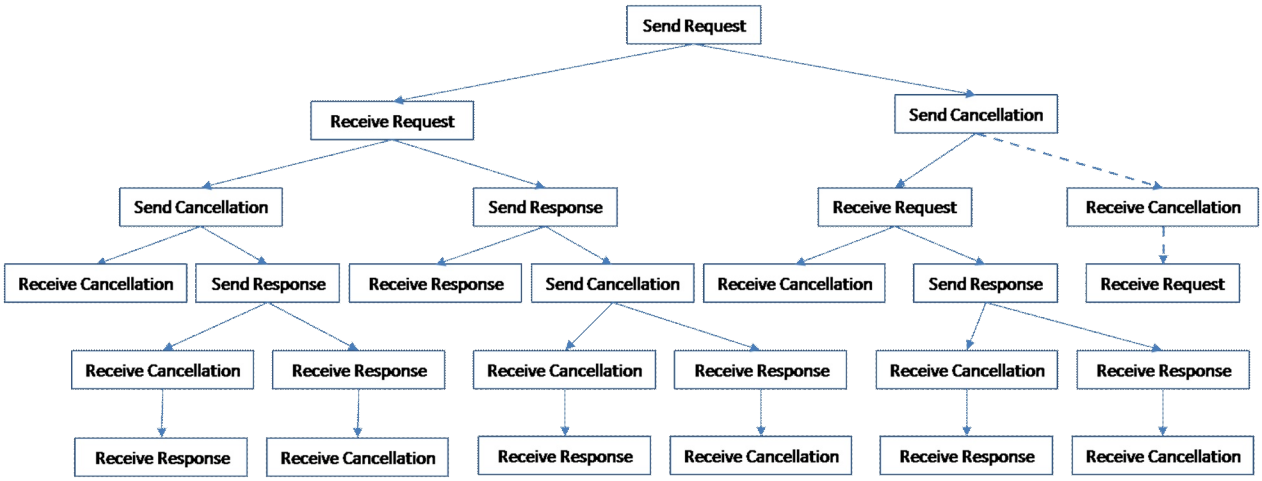


Figure 4. All possible traces of the composed system in Figure 3

disjoint subsets *MT'* of *MT* to channel *reliability guarantees*[2] which enforce that $\pi_{MT'}(r)$ satisfies certain properties. Reliability guarantees such as those from WS-RM standard [14] can be modeled:

- *exactly once in order (EOIO)* where $\pi_{MT'}(r)$ is the identity function on message sequences and
- *exactly once (EO)* where $\pi_{MT'}(r)$ is a permutation on a message sequence.

Weaker guarantees, where message loss, modification, or duplication can occur are treated at a lower level in the technology stack, so we do not consider them in MCM. In practice, EO is preferred over EOIO, since it is less costly to enforce.

3.4 Composed System

A *composed system* consists of a set of LPMs and a channel model over some common set of message types. Its state space is defined by the notion of *composed state*, consisting of

- a local state of each LPM and
- a sequence of messages of the system's message types currently sent but not yet received (that means they are on the channel).

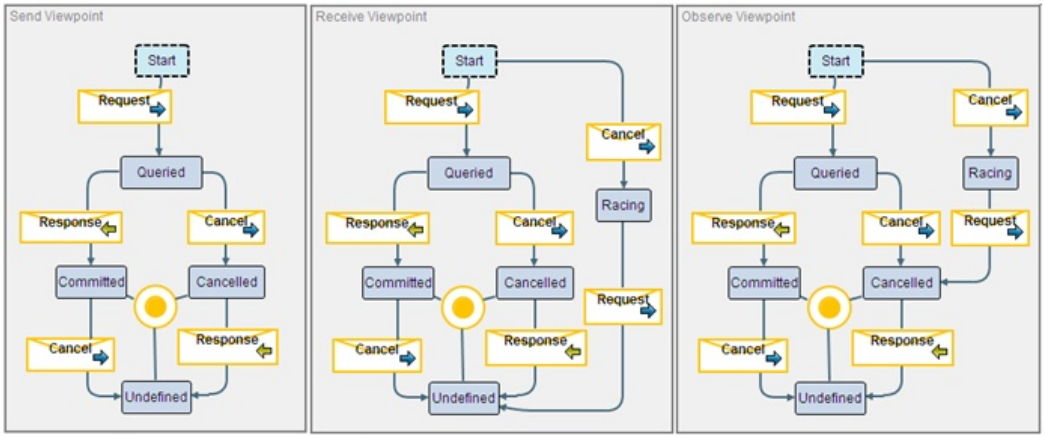
Given a composed system *L*, its labeled transition system denoted by *LTS(L)* is defined on the composed states and the transitions labeled by send and receive events of the LPMs. The transitions are described in steps 2-4 below. The behavior of *L*, denoted by *Traces(L)*, is defined by the sequences $(e_1,\ldots,e_n)$ of send or receive events e_k $(k=1,..,n)$ of the LPMs involved in the composed system which satisfy the following property. There is a sequence $(s_0,\ldots,s_n)$ of composed states such that for all e_i $(i=1,..,n)$, its message type *mt*, and its partner *p*, the following holds:

1. In s_0 the channel is empty and all LPMs are in their initial local states (i.e. composed initial state).
2. The state of *p* in s_{i-1} satisfies the guard condition of e_i and the state of *p* in s_i equals the effect of e_i and s_{i-1} equals s_i in all other partners' states.
3. If e_i is a send event, then in s_i the channel sequence equals the channel sequence of s_{i-1} prepended with a new message of *mt*.
4. Let *r* be the reliability guarantee assigned to the subset *MT* of message types with *mt* in *MT*. Let further be *ch* the sequence of messages on the channel in s_{i-1}. If e_i is a receive event, then with $r(ch)=(m_1,\ldots,m_k)$:
   - m_k is of type *mt* and
   - the channel sequence of s_i equals $r(ch)$ without its last element m_k.
5. In s_n all reached local states are target states and the channel is empty (i.e. composed target state).

For example, the global trace (*Request*, *Response*) and its associated state sequence (*Start*, *Queried*, *Commited*) correlates to the composed trace (*Send Request*, *Receive Request*, *Send Response*, *Receive Response*) and its associated composed state sequence:

({*Buyer.Start, Seller.Start, Channel* = ()},
{ *Buyer. Queried, Seller.Start, Channel* = (*Request*)},
{ *Buyer. Queried, Seller. Queried, Channel* = ()},
{ *Buyer. Queried, Seller.Commited, Channel* = (*Response*)},
{ *Buyer.Commited, Seller.Commited, Channel* = ()}).

[2] In the context of SAP applications, it is common to assign reliability guarantees per message type for the communication between two components.



(a) (b) (c)

Figure 5. Global choreography models reflecting the three message exchange viewpoints

For the composed system for the LPMs in Figure 3 and an EO channel assumption, the resulting traces are depicted in Figure 4. For an EOIO channel, the dashed path is excluded because the request message cannot overtake the cancellation message.

3.5 Advanced Features of MCM

The modeling concepts introduced until now cover a large part of the choreographies we have to deal within industrial practice. However, MCM has a couple of advanced features sketch below, which were required by the use cases that we investigated. Sometimes in practice, the available pre-defined message types are not expressive enough. For instance, an interaction with a confirmation message type may either express an acceptance (i.e., positive confirmation) or a rejection (i.e., negative confirmation) and thus the communication might continue in different ways, depending on this information. Therefore it is necessary to take into account the contents of the exchanged messages as additional guards of interactions. Otherwise, non-determinism would impede generation of integration tests. In MCM, we have added a message constraint language consisting of first-order expressions over data type properties. For example, we refer to contents of an exchanged message *msg* by:

msg[PurchaseOrder.AcceptanceStatusCode]=="accepted"

expressing that the contents of the message by navigating through *PurchaseOrder* and *AcceptanceStatusCode* has a certain value that enables the interaction.

Moreover interaction guards may refer to messages exchanged in other (previous) interactions or to a global counter of messages. Therefore we introduced additional variables having an assigned data type. These variables can be declared globally, assigned to in interactions, and referred to in guards.

A further improvement comes with considering the global state to be assembled of a number of parallel sub state machines. Partitioning the global state in this way allows explicit concurrency modeling which can concisely reflect the inherent asynchronous nature of choreographies.

4. Message Exchange Viewpoints

Given a fixed set of partners and message types, we assume to have a GCM *G* and composed system *L* over the given set of partners. In Sections 3.1 and 3.4 we associated *G* and *L* with their labeled transition systems *LTS(G)* and *LTS(L)* as well as their traces *Traces(G)* and *Traces(L)*. Note that while *LTS(G)* has a finite number of states, *LTS(L)* may have an infinite number of states if there are no restrictions on the channel size.

Note that the GCM *G* and the composed system *L* have different alphabets. Therefore we need to map the alphabet of interactions used by *G* to their corresponding send and receive events in *L*, in order to define a consistency relation between them. How this is done is determined by the three possible viewpoints. We explain them in detail using our example.

For simplicity, we fix the consistency relation between *G* and *L* to trace inclusion and assume *G* to be an over-approximation of *L*, that is the set *Traces(L)* is included in the set *Traces(G)*. Other variants for the consistency relation between *G* and *L* may be considered (see Section 0). Further let *MT(m)* denote

the message type of an interaction as well as of a send or a receive event *m*.

Send viewpoint. A GCM *G* is a *send-viewpoint* of a composed system *L* if for each trace $(e_1,\ldots e_n)$ of *L* there exists a trace $(i_1,\ldots,i_k)$ of *G* such that if $(s_1,\ldots s_k)$ is the projection of $(e_1,\ldots e_n)$ to send events, then $(MT(i_1),\ldots,MT(i_k))= (MT(s_1),\ldots MT(s_k))$.

The GCM in Figure 5(a) is a send-viewpoint of the composed system depicted in Figure 3 with one EO channel for all messages. The projection of *Traces(L)* into send events is the set:

send request → send cancel
send request → send response
send request → send cancel → send response
send request → send response → send cancel

This corresponds to the traces of the considered GCM. Note that taking into account or omitting the additional dashed "EO trace" does not affect the resulting traces above. Thus the send viewpoint does not reflect message racing explicitly. Its choreography model is equal to the one describing ordered messages in Figure 2.

Receive viewpoint. A GCM *G* is a *receive-viewpoint* of *L* if for each trace $(e_1,\ldots e_n)$ of *L* there exists a trace $(i_1,\ldots,i_k)$ of *G* such that if $(r_1,\ldots r_k)$ is the projection of $(e_1,\ldots e_n)$ to the receive events, then $(MT(i_1),\ldots,MT(i_k))= (MT(r_1),\ldots MT(r_k))$.

The GCM in Figure 5(b) is a receive-viewpoint of the composed system from Figure 3 with an EO channel, because the projection of the traces from Figure 4 to receive events results in

receive request → receive cancel
receive request → receive response
receive request → receive cancel → receive response
receive request → receive response → receive cancel
receive cancel → receive request

This corresponds to the traces of the global model in Figure 5(b). If we consider an EOIO channel for all messages the dashed path is omitted and thus the last of the above receive-projected traces is excluded. Hence all the models in Figure 5 are receive-viewpoints of the composed system with EOIO.

The receive-viewpoint thus reflects the possible loss of message order on the channel in defining all possible sequences of receive events. In our example the buyer's *cancel* message can possibly overtake the *request* message, but if received in that order the seller will not send any *response*.

Observe viewpoint. We can consider an even weaker viewpoint of GCMs. Assume that we *extend* the traces of a composed system with an *observe* event, similarly to send- and receive-events. It can occur at any position between the send- and the receive-event of a particular message. E.g., if *(send_m1, send_m2, receive_m1, receive_m2)* is a composed system trace then

{(send_m1, ***observe_m1****, send_m2,* ***observe_m2****, receive_m2, receive_m1),*
(send_m1, send_m2, ***observe_m1****,* ***observe_m2****, receive_m2, receive_m1),*
(send_m1, send_m2, ***observe_m2****,* ***observe_m1****, receive_m2, receive_m1),*
(send_m1, send_m2, ***observe_m2****, receive_m2,* ***observe_m1,*** *receive_m1)}*

is an *observe-extended trace*.

Similar to previous definitions, a GCM *G* is an *observe-viewpoint* of a composed system *L* if for each observe-extended trace $(e_1,\ldots e_n)$ of *L* there exists a global trace $(i_1,\ldots,i_k)$ such that if $(o_1,\ldots o_k)$ is the projection of $(e_1,\ldots e_n)$ to the observe events, then $(MT(i_1),\ldots,MT(i_k))= (MT(o_1),\ldots MT(o_k))$.

In our example, *observe cancel → observe request → observe response* is an additional (observe) trace of the composed system with EO channel. The global model of Figure 5(c) is thus an observe-viewpoint of the composed system in Figure 3, but those in Figure 5(a) and Figure 5(b) are not.

The presence of the additional trace is due to the fact that there might be a point of observation where the messages *cancel* and *request* from the buyer switch their order but switch back to the original order before reaching the seller. In this case the seller might respond to the *request* message before receiving the *cancel*.

Discussion. Existing approaches in the literature (see Sect. 7) do not disambiguate between different viewpoints, thus leaving room for interpretation. When applicable, the approaches implicitly assume a send-viewpoint. We found however that a send-viewpoint (though applicable) is probably misleading as it does not reflect the possibility of message order changes. The same holds for an observe-viewpoint as it describes message sequences, that the two participating partners will not observe locally. In contrast, the receive-viewpoint exhibits exactly those orders of receive events that are possibly observed by the partners. This is important for test generation which is supposed to uncover also message racing problems. Moreover, if we considered using the GCM for monitoring purposes like runtime testing, then the observe-viewpoint would have to be chosen, as it reflects possible message racing between the components and an observer too.

Note that if one considers EOIO for all messages, the three viewpoints are equivalent. However, already when there are two EOIO channels for messages in the same direction the equivalence does not hold anymore.

5. Global vs. Local Viewpoints

In this section we focus on the issue that choreographies can be considered from the perspective of a global observer and from that of the individual partner components. From a practitioner's perspective both have their right to exist: While the first is a most concise description of the service interaction, the latter is closer to the component implementation and is a more suitable starting point to verify the components.

MCM offers models for both perspectives: GCM and LPMs. Below we present how these two viewpoints are kept in synchronization.

In the literature, two competing approaches exist for this task: a generative approach where the local views are generated from the global ones, or a checking approach where global and local models are created separately and then verified whether they are consistent with each other. While the first ensures that global and local views are always consistent, it makes changes to the local models considerably more difficult, since these would be overridden by re-generation from the global model. The latter approach allows for such "asymmetric" changes, but requires manual effort to update the global view when changes to the local models are made.

We suggest a third course of action which attempts to combine these approaches, by having a single metamodel for both GCM and LPM. Thus, for each choreography, GCM as well as the two LPMs for each involved partner are views on one common instance of that metamodel. For each partner, an LPM view is obtained from a GCM as follows. An interaction in the GCM is interpreted as a send or as a receive event in the LPM of the considered partner (depending on whether that partner sends or receives the associated message according to the PCIM). A state in the GCM is represented by a corresponding state in the LPM, and constraints and effects are accordingly transferred.

Following this approach, the addition of a send-event to one of the LPMs automatically leads to the addition of an interaction in GCM and to the addition of a receive-event to the other partner's LPM.

Note that, semantically, the GCM states and the corresponding LPM states are different: The former denotes a globally observed state (based on the chosen message exchange viewpoint), while the latter denotes the latest information that the components have about the global state. These states are in general not equal, because of the latency of message transmission.

By this technique, we obtain the most general LPMs which realize a choreography given by a GCM. In order to allow for asymmetric resolution strategies, the LPMs can be augmented with additional guards. As discussed in [13], the added guards can only restrict that messages are sent. These additional constraints are however only visible in the particular LPM, but not in GCM nor in the LPM of the other partner.

Our approach thus ensures a "syntactical" consistency of global and local viewpoint during the whole modeling lifecycle, but allows for (restricted) changes to only one of the involved components. Still it is possible at this stage that the GCM *G* and composed system *L* of LPMs are inconsistent.

Consistency between the local and global views. There are various types of consistencies between *G* and *L*. One may consider the simulation relation between *LTS(G)* and *LTS(L)* or trace inclusion of *Traces(G)* and *Traces(L)*. Moreover, one can check if *G* is an over-approximation, under-approximation or equivalent to *L*. In the literature, for instance, the *local enforceability* from [4] corresponds in our setting to *G* as over-approximation of *L* based on simulation, while the *realizability* from [4] corresponds in our setting to the bisimulation between *G* and *L*. A send viewpoint is assumed there. While our framework covers these cases, we further address consistency relations based on under-approximations and a receive viewpoint. For instance, a receive viewpoint solves the shortcoming mentioned in [3] where the authors conclude that weak bi-simulation does not support reordering of messages. The message exchange viewpoints that we introduced can also be used to sharpen verification based approaches taking asynchronous messaging into account, see for example [6].

Our MCM tool provides checks which prove consistency, based on translating the choreography to Event-B and checking the obtained formal model with Rodin platform tools [9].

6. MCM in the Development Process

In this section we explain the development activities that involve the use of MCM models.

Initial modeling. The aim of building the GCM is to create a description that fits all the user

requirements and at the same time does not violate basic desirable properties like deadlock-freedom and local enforceability [4]. Usually for complex interactions between services such descriptions have to be created by stepwise refinement. Therefore we start with the definition of an initial GCM that might not be complete and usually considers synchronous communication for simplicity. From the GCM, the editor automatically derives the LPMs (see previous section). In the next step, these LPMs are manually supplemented with local send-after-receive constraints. Finally, the channel reliability is defined for each message type (see Sect. 3.3).

Verification and MCM refinement checks. After the initial model is built, it can be used for verification. Our implementation uses the general purpose formal modeling language Event-B and the tool Rodin [9]. One of the central concepts of Event-B is refinement, which covers both trace inclusion and simulation. Therefore it can be used for checking all the consistency relations from Sect. 0. In addition to the consistency relations, we also verify properties like absence of inconsumable messages and deadlocks. Usually enhancements of the GCM and restrictions of the LPMs and channel models both are carried out in one refinement step. After each refinement, the verification will be applied again until all user requirements are modeled and the checks do not uncover issues any more.

Test generation. Because the created models are too abstract to generate code automatically, the implementation has to be carried out manually. For the consecutive integration testing however the granularity of MCM is fine enough so that model-based testing techniques can be applied [11]. On one side choreography models can be used to generate test suites that satisfy chosen coverage criteria. On the other side the models can serve as a test oracle. Both methods are combined in our implementation, so that the generated test cases contain information about the expected system response that is compared against the occurring output. The generated abstract test suites are transformed into concrete test suites by adding test data and further into executable test scripts by providing system specific information.

7. Related Work

As mentioned in Section 1, the differentiation between static and behavioral viewpoints when describing service compositions is common knowledge in literature. For static descriptions, WSDL has become widely accepted. For dynamic descriptions, several choreography modeling languages have emerged in the last few years. Some of the most prominent languages are WS-CDL [8], BPMN [1], BPEL4Chor [2], and Let's Dance [16]. They vary in several regards such as abstraction level, formal grounding, target users, etc.

WS-CDL is a choreography language that targets the implementation level and builds on WSDL. Its lack of a graphical syntax decreases its usability for modelers. It also misses the explicit notion of termination, which is an important ingredient for test generation. Termination can only be modeled by having no further possibility to send any message and hence the target states *Committed* and *Cancelled* in Figure 2 cannot be modeled in WS-CDL without leading to non-determinism. WS-CDL further assumes a global send viewpoint and hence does not reflect message racing directly.

BPMN 1.1 is a language to describe process modeling not targeting choreography modeling. The recent draft for BPMN 2.0 [1] explicitly includes choreography modeling that should be understandable by business users and technical developers alike. In its current state BPMN 2.0 has a too restrictive notion of termination. Even though (unlike WS-CDL) end states are defined explicitly, they do not contain outgoing transitions and thus lead to the same problems described above. Further it abstracts from channel information and does not define message exchange viewpoints. Moreover BPMN 2.0 has a large number of (non-trivial) modeling artifacts that makes the modeling process complex and the learning curve steep.

BPEL4Chor and Let's Dance focus on high-level service interaction modeling in early design phases and target business analysts. Although the core of these languages is formal, guards and conditions can only be defined in natural language, which makes them inappropriate for verification or automatic testing approaches. Like WS-CDL they do not have explicit notion of termination. Due to the assumed send viewpoint they are not able to visualize message racing. Additionally, Let's Dance does not support the modeling of partner views.

We concluded that the semantics of all these languages would have to be refined (e.g. viewpoints, termination, determinism enforcement, channel restrictions) to suit our purpose. Moreover, the implementation of a supporting tool infrastructure would be necessary. Therefore it seemed more appropriate to define a domain specific language like

MCM that fully fits the intended use and eliminates the arising semantic uncertainties of unspecified viewpoints.

A common metamodel that covers the global and local viewpoints has also been introduced in [5]. The global choreography model even depicts events that occur between send and receive and hence implicitly assumes an observe viewpoint. However the modeling of different channels is not a focus of this work and hence the message exchange viewpoints cannot be applied directly.

8. Conclusions

In this paper we presented a choreography modeling language called MCM that fulfills the requirements SOA poses on choreography modeling in an ERP software development context [13], e.g. a detailed channel model, connected global and partner models, a well-defined viewpoint for the global model and the availability of termination symbols. Current approaches are not designed to tackle all of these concerns simultaneously. Currently our approach is evaluated at selected SAP development groups, so far with very positive feedback. This was due to the tight integration of MCM with existing SAP metamodels.

We introduced a variety of options for interpreting global choreography models and for defining consistency with local models, which have not explicitly been addressed in literature. Especially the three message exchange viewpoints based on the send, receive, and observe events help to sharpen the semantics of choreographies considerably. We found different use cases for the three different viewpoints: the send viewpoint is suitable for service design, the receive viewpoint for testing service integration, and the observe viewpoint for service monitoring.

While this paper mainly focused on the MCM and different choreography viewpoints, upcoming papers will give more details about the testing [11] and verification methods that we developed.

Acknowledgments. This work was partially supported by the EC-funded projects MODELPLEX, Deploy, and VIDE (grants no. 034081, 214158, and 033606).

9. References

[1] Business Process Modeling Notation (BPMN) Specification 2.0, Submitted Draft Proposal V0.9. Online at: *http://www.omg.org/cgi-bin/doc?bmi/08-11-01*

[2] Decker, G., Kopp, O., Leymann, F., Weske M.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: International Conference on Web Services Oriented Computing (ICWS'07), pp. 296–303. IEEE Computer Society (2007)

[3] Decker, G., Weske, M.: Behavioral Consistency for B2B Process Integration. In: Proc. of CAiSE'07. LNCS, vol. 4495, pp. 81–95. Springer (2007)

[4] Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: Proc. of BPM'07, LNCS, vol. 4714, pp. 305–319. Springer (2007)

[5] Dijkman, R., Dumas: Service-Oriented Design: A Multi-Viewpoint Approach. In: Int. J. Cooperative Inf. Syst. 13(4), pp. 337–368. World Scientific (2004)

[6] Kazhamiakin, R., Pistore, M.: Choreography Conformance Analysis: Asynchronous Communications and Information Alignment. In: Proc. of WS-FM'09, LNCS, vol. 4184, pp. 227-241. Springer (2006)

[7] Kätker, S., Patig, S.: Model-driven Development of Service-oriented Business Application Systems. In: Business Services: Konzepte, Technologien, Anwendungen, 9. Internationale Tagung Wirtschaftsinformatik. Band 1, pp. 171-180. Österreichische Computer Gesellschaft (2009)

[8] Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Candidate Recomm., Technical report (2005)

[9] Métayer, C., Abrial, J.R., Voisin, L.: Event-B Language, Online at: *http://rodin.cs.ncl.ac.uk*

[10] O'Leary, D.E.: Enterprise Resource Planning Systems - Systems, Life Cycle, Electronic Commerce And Risks. Cambridge University Press (2000)

[11] Stefanescu, A., Wieczorek, S., Kirshin, A.: MBT4Chor: A Model-based Testing Approach for Service Choreographies. In: Proc. of ECMDA'09, LNCS, vol. 5562, pp. 313–324. Springer (2009)

[12] van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In: Proc. of CAiSE'01. LNCS, vol. 2068, pp. 140–156. Springer (2001)

[13] Wieczorek, S., Roth, A., Stefanescu, A., Charfi, A.: Precise Steps for Choreography Modeling for SOA Validation and Verification. In: International Symposium on Service-Oriented Software Engineering (SOSE'08). IEEE Computer Society (2008)

[14] Web Services Reliable Messaging (WS-Reliable-Messaging), Vers. 1.1. OASIS Consortiom. Online at: *http://docs.oasis-open.org/ws-rx /wsrm/v1.1/wsrm.pdf*

[15] Woods, D., Mattern, T.: Enterprise SOA – Designing IT for Business Innovation. O'Reilly (2006)

[16] Zaha, J. M., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: Proc. of CoopIS'06, LNCS, vol. 4275, pp. 145–162. Springer (2006)