Let's Dance: A Language for Service Behavior Modeling

Johannes Maria Zaha[1], Alistair Barros[2], Marlon Dumas[1], Arthur ter Hofstede[1]

[1] Queensland University of Technology, Brisbane, Australia
(j.zaha,m.dumas,a.terhofstede)@qut.edu.au
[2] SAP Research Centre, Brisbane, Australia
alistair.barros@sap.com

Abstract. In Service-Oriented Architectures (SOAs), software systems are decomposed into independent units, namely services, that interact with one another through message exchanges. To promote reuse and evolvability, these interactions are explicitly described right from the early phases of the development lifecycle. Up to now, emphasis has been placed on capturing structural aspects of service interactions. Gradually though, the description of behavioral dependencies between service interactions is gaining increasing attention as a means to push forward the SOA vision. This paper deals with the description of these behavioral dependencies during the analysis and design phases. The paper outlines a set of requirements that a language for modeling service interactions at this level should fulfill, and proposes a language whose design is driven by these requirements.

1 Introduction

As the first generation of web service technology based on XML, SOAP, and WSDL, reaches a certain level of maturity and adoption, a second generation based on richer service descriptions is gestating. Whereas in first-generation web services, service descriptions are usually equated to sets of operations and message types, in the second generation the description of behavioral dependencies between service interactions (e.g. the order in which messages must be exchanged) plays a central role.

Two standardization initiatives, BPEL [1] and WS-CDL [3], have promoted the description of behavioral aspects of service interactions. However, these proposals focus on the implementation phase of the service development lifecycle. Indeed, although WS-CDL claims to be at a higher level than BPEL, they both rely on procedural programming constructs such as variable assignment and instruction sequencing. One can argue that during analysis and design, capturing constraints on possible interactions is more relevant than prescribing interaction procedures. Furthermore, while BPEL focuses on describing interactions from a local perspective, i.e. from the perspective of a specific service, WS-CDL emphasizes global descriptions (also called choreographies).

Previous proposals have defined extensions of well-known behavior modeling paradigms (e.g. Activity Diagrams) to capture message exchanges in the style of service interactions. This is the case for example of BPMN [19] and BPSS (or ebBP) [6]. However, these languages do not treat service interactions as first-class citizens but rather as extensions to a core language centered around the notion of actions and dependencies between actions. In particular, service interactions in these languages are not composable, meaning that it is not possible to aggregate several interactions and treat them as a single interaction to which it is possible to apply the same operators as for elementary interactions. It is thus questionable whether these languages meet the requirements of a language for service interaction modeling. Also, these languages do not provide a unified framework for capturing interactions both from a local and from a global viewpoint.

In prior work, we have captured suitability (i.e. fit-to-purpose) requirements for service interaction languages and documented them in the form of 13 patterns [2]. In this paper, we formulate additional requirements for a service interaction modeling language and use these as the basis for a language proposal. The main contribution of the paper is thus a language, namely "Let's Dance", for modeling behavioral dependencies between service interactions. The language is targeted at business analysts and software architects involved in the initial phases of the service development lifecycle. Accordingly, it abstracts away from implementation details and avoids reliance on imperative programming constructs. At the same time, models defined in the proposed language contain information that can be leveraged upon during the implementation and operations phases, for example to generate BPEL templates or to ensure, through monitoring or log analysis, that a given service implementation conforms to the requirements expressed in the models.

The paper is structured as follows. In Section 2 the requirements for a service behavior modeling language are formulated. Section 3 gives an overview of the Let's Dance language. In Section 4 the suitability of the language is illustrated through scenarios corresponding to some of the interaction patterns of [2]. Section 5 introduces the meta model of the language as well as its informal semantics. Finally, Section 6 concludes and gives an outlook on future work. A detailed evaluation of the language with respect to the service interaction patterns is given in appendix.

2 Requirements and Related Work

The intended scope of the Let's Dance language is to capture models of service interactions from a behavioral perspective. The language is targeted at the analysis and design phases of the systems development lifecycle. As such, it should be sufficiently *abstract* (i.e. *conceptual*), meaning that it should allow modelers to focus on the essence of their service interaction analysis and design problems, abstracting away from implementation details. Accordingly, we aim at defining a language that is as free as possible from programming constructs such as explicit

variable assignment. Examples of what the language should not force modelers to do include:

- Having to intersperse variable assignment actions for book-keeping purposes, e.g. using variables as counters or as "flags" to indicate that a state has been reached, or as "buffers" to store the result of intermediate computations.
- Having to introduce unnecessary ordering or synchronization constraints, i.e. sequentializing interactions that could otherwise be performed in any order or in parallel.

A major use case of the language is to define models of service interactions that can be refined into implementations, or compared against the behavior exhibited by an existing implementation. Thus, the language should have an unambiguous semantics and it should be possible to compare the semantics of a model with the behavior exhibited by an implementation. In addition, it is desirable to reason about the models defined in the language, e.g. to verify or test certain properties through static analysis or simulation. One way to achieve these requirements is to endow the language with a *formal* and *executable* semantics.

Another major purpose of the language is to facilitate communication between analysts, designers, and other stakeholders involved in the development of service-oriented systems. Users of the language, including non-technical users, will need to understand, critique, modify or suggest modifications to models or parts thereof. The language must hence be *comprehensible*. This comprehensibility can be achieved by: (i) attaching a graphical syntax to the language to better exploit the perceptual capabilities of users, and (ii) allowing models to be captured at different levels of detail and from different viewpoints. In particular, the language should allow interactions to be decomposed into other interactions so that users can choose the level of details at which they wish to view an interaction model. Also, users should be able to design and view models from a local and from a global perspective. In the global (or choreography) perspective, interactions are described from the perspective of a collection of services (abstracted as roles). This is useful when communicating about how services should behave in order to seamlessly interact with one another. On the other hand, local models focus on the perspective of either an existing or a "to be" service, capturing only those interactions that involve that particular service. A possible usage scenario is one where global models are produced by analysts to agree on major interaction scenarios, while local models are produced during system design and handed on to implementers. To ensure proper handovers between these users, it is necessary to have a mapping from global to local models and/or to be able to check that a local model is consistent with a global one. In other scenarios, reverse mappings from local to global models may also be useful.

Finally, the language must be *expressive* and *suitable*. Given the intended scope, expressiveness refers to the ability to capture any set of service interactions and their associated behavioral dependencies. Suitability on the other hand, refers to the ability to capture common scenarios in an intuitive manner. Languages such as Colored Petri nets [13] or Live Sequence Charts [7] allow one to capture any service interaction scenario (in computational terms, they

are Turing complete). However, they are not necessarily suitable for the task at hand: modeling certain common interaction scenarios would require the use of programming constructs. So while expressiveness is certainly important and we plan to investigate this in the future, our initial language design has been driven by suitability. Suitability is arguably a subjective property and is ultimately determined by the users and the use cases. We have chosen to take the service interaction patterns proposed in [2] as a way of evaluating suitability. These patterns have been derived from insights into real-scale B2B transaction processing, use cases gathered by standardization committees (e.g. BPEL and WS-CDL) during their requirements analysis phase, and scenarios identified in industry standards (e.g. xCBL choreographies and RosettaNet PIPs [18]). The proposed patterns, as such, are not complete but aim at consolidating recurrent scenarios, abstracting them in a way that provides reusable knowledge to service developers. Since the patterns are abstractions of recurrent scenarios, they can be used to assess the suitability of a given language for service behavior description. Accordingly, a driving requirement of our language design is that it should provide direct support for these patterns. Section 4 and the appendix shows how each of these patterns are captured in Let's Dance.

Existing languages for capturing service interaction behavior include WS-CDL [3], BPEL [1], BPMN [19], UML Activity Diagrams [16] (and variants thereof as defined in related initiatives such as BPSS [6] or EDOC [15]). WS-CDL and BPEL are aimed at the detailed design and implementation stages of the development lifecycle. They both extensively rely on programming language constructs such as sequence, "repeat until" and "while" loops, and variable assignment. Capturing service interaction patterns related to multi-party or streamed interactions in these languages requires extensive use of variables for record-keeping purposes.[3] Also, BPEL does not allow one to represent global views. Moreover, these languages do not have a graphical syntax, although different tools assign them various graphical representations. UML activity diagrams and BPMN do have a prescribed graphical representations and are targeted at analysts and designers. However, in these languages interactions are not treated as first-class citizens. Elementary interactions (i.e. message exchanges) can be represented through "message flows" and "object flows" respectively. However, these constructs are not composable: It is not possible to represent an interaction that is composed of other interactions. And as in the case of BPEL and WS-CDL, UML and BPMN do not provide constructs to capture multi-party and streamed interactions and these would need to be encoded using loops and variable manipulation for record-keeping.

Another family of languages that have been proposed for capturing service interactions are the so-called "semantic web service" description languages. This family of languages includes OWL-S [14] and WSMO [17]. The basic idea of these languages is to use logical statements to capture and manipulate service-related information. In these languages, a service is described as a set of facts and rules covering three broad aspects:

[3] See sample code available at `www.serviceinteraction.com`.

– Capability: what can the service do?
– Non-functional properties: conditions of usage, contractual terms, contextual constraints, etc.
– Interface: preconditions, post-conditions and/or effects of each interaction in which the service can engage. Interface descriptions are akin to local models in Let's Dance, although they cover both structural and behavioral aspects.

While semantic web service descriptions are suitable in view of applying automated reasoning techniques (e.g. automated planning) over service descriptions, their suitability for use at the level of domain analysis and systems design is questionable. Domain analysts do not typically describe services down to the level of details required for non-trivial automated reasoning. Thus, semantic web service description languages can be seen as possible target languages for Let's Dance. For example, Let's Dance local models could be used to generate templates of WSMO descriptions, which after refinement, could be given to automated reasoning engines to determine if a service can be combined with other services to achieve a given goal.

In [4], Finite State Machines (FSMs) are put forward as a suitable language for modeling service interactions. However, while state machines lead to simple models for highly sequential scenarios, they may lead to complex, spaghetti-like models when used to capture scenarios with parallelism and cancellation (e.g. scenarios where a given interaction may occur at any time during the execution of another set of interactions).

Foster et al [10] suggest the use of Message Sequence Charts (MSCs) for describing global service interaction models. These global models are converted into local models expressed as FSMs for analysis purposes. It should be noted that MSCs are a notation for describing behavior scenarios as opposed to full behavior specifications. In particular, basic MSCs do not allow one to capture conditional branches, parallel branches, and iterations. Extensions to MSCs to capture complex behavior have been defined, but in realistic cases they lead to cluttered diagrams since MSCs are based on lifelines which are fundamentally targeted at capturing sequencing rather than branching.

3 Language Overview

Due to the variety of constructs that can be used to build models, the identification of the best way to depict a given issue is nearly impossible. We have used the cognitive dimensions for visual programming environments introduced in [11] as a guideline for optimizing the choice of constructs for the visual syntax of Let's Dance. Note that this optimization can only be done along one dimension and there are usually trade-offs between the different optima of each dimension [5]. For the design of a graphical modeling language, important dimensions to consider, among those presented in [11], are abstraction gradient, consistency and diffuseness. The abstraction gradient covers the minimum and maximum levels of abstraction that must be captured and the possibility to encapsulate fragments of a certain model. In a language for service behavior modeling there are two

levels of abstraction: the global model is the higher level while the local model is the lower one. Closer inspection shows that these levels can themselves be described at different levels of abstraction and this can be achieved by ensuring composability. The consistency or harmony of a language testifies how much of a language has to be learnt by a user in order to infer the rest. This inference should be easy for a language fulfilling the above requirements, since the same constructs ought to be used for describing local and global models. Finally, the dimension of diffuseness reflects how many symbols or graphic entities are required to express a given issue. This dimension is determined by the need to support local and global models and by the concepts involved in each of these models.

Service interactions can be described in terms of message exchanges. A message exchange consists of a message sending and a message receipt. Thus, at the lowest level of abstraction, a language for modeling service behavior must provide these two constructs. To optimize the abstraction gradient and consistency dimensions discussed above, we choose a notation wherein the visual juxtaposition of the symbols for sending and receiving messages leads to the symbol for elementary interactions (i.e. message exchanges). Interactions can then be related and composed to form choreographies. Communication actions are represented by the non-regular pentagons shown in the left-hand side of Figure 1. As illustrated in this figure, we distinguish between a message sending that requires an acknowledgment and one that does not. Similarly, we distinguish between message receipt actions that provide acknowledgment and those that do not.[4]

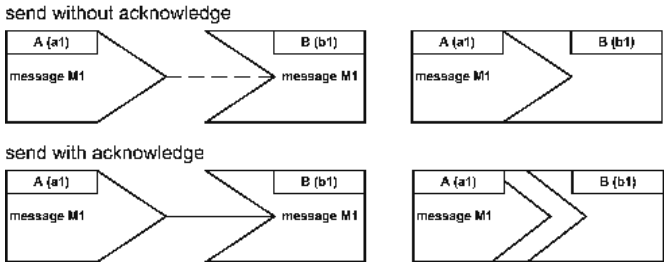

Fig. 1. Communication Actions and Interactions

A communication action is performed by an actor playing a role. This information is specified at the top corner of the pentagon denoting a communication action. Roles are written in uppercase and the actor playing this role (specifically, the "actor reference") is written in lowercase between brackets. The symbol for sending and receiving messages are combined to form elementary interactions by

[4] At the modeling level, we are only concerned with capturing whether an acknowledgment is needed or not. The protocol used for acknowledging is an implementation concern.

juxtaposing the respective symbols (see right-hand side of Figure 1). Again, we distinguish between elementary interactions with and without acknowledgment.

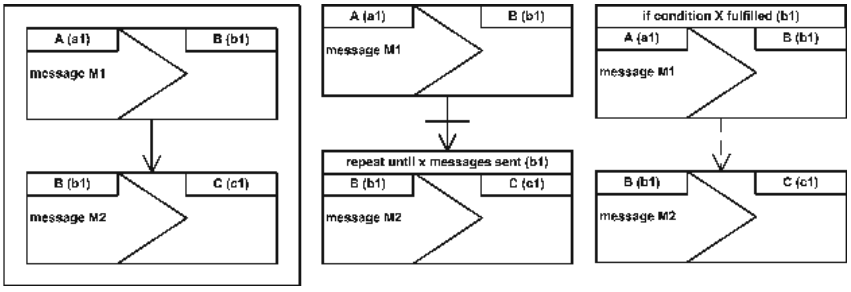

Fig. 2. Relationships between interactions

Interactions can be inter-related using the constructs depicted in Figure 2. The relationship on the left-hand side is called "precedes" and is depicted by a directed edge: the source interaction can only occur after the target interaction has occurred. That is, after the receipt of a message "M1" by "B", "B" is able to send a message "M2" to "C". The rectangle surrounding these two interactions denotes a composite interaction, which can be related with other interactions with any type of relationship. The sub-interactions of a composite interaction may, but need not be related. If there is no relationship between them, they can occur in any order or in parallel.

The relationship at the center of the figure is called "inhibits", depicted by a crossed directed edge. It denotes that after the source interaction has occurred, the target interaction can no longer occur. That is, after "B" has received a message "M1" from "A", it may not send a message "M2" to "C". The latter interaction can be repeated until "x" messages have been sent, which is indicated by the header on top of the interaction. The actor executing the repetition instruction is noted in brackets. Additional to the "Until" construct, Let's Dance provides two more constructs to denote the repetition of an interaction, namely "For each" and "While".

Finally, the relationship on the right-hand side of the figure, called "weak-precedes", denotes that "B" is not able to send a message "M2" until "A" has sent a message "M1" or until this interaction has been inhibited. That is, the target interaction can only occur after the source interaction has reached a final status, which may be "completed" or "skipped" (i.e. "inhibited"). In the example, the upper interaction has a guard assigned, which is denoted by the header on top of the interaction. Guards allow for the definition under which condition a given interaction can be executed and can be assigned to any interaction, whereby the actor evaluating a guard has to be named explicitly.

All these constructs will be exemplified in the following section, where we evaluate the suitability of Let's Dance with respect to the interaction patterns of [2].

4 Interaction Patterns in Let's Dance

The service interaction patterns introduced in [2] have been put forward as an instrument for benchmarking languages for service behavior modeling. In [2], solutions to the patterns in BPEL are sketched (the full solutions can be found in `www.serviceinteraction.com`). These patterns are divided in four categories: single-transmission bilateral interaction patterns, single-transmission multilateral interaction patterns, multi-transmission interaction patterns and routing patterns.

We have assessed the suitability of Let's Dance by modeling sample scenarios corresponding to all 13 patterns. This section only considers representative patterns of each of the four categories. A discussion on how Let's Dance addresses the remaining patterns is given in appendix. Using the nomenclature and numbering of [2] we have chosen the following patterns: Send/Receive (pattern 3), Racing incoming messages (pattern 4), One-to-many send/receive (pattern 7), Multi-responses (pattern 8) and Relayed Request (pattern 12). For each of the patterns, its description and a model corresponding to one of the sample scenarios given in [2] are provided. Each scenario is then modeled in Let's Dance and the resulting models are informally discussed.

Send/Receive. The Send/Receive pattern is described as follows [2]: "A party X engages in two causally related interactions: in the first interaction X sends a message to another party Y (the request), while in the second one X receives a message from Y (the response)." The example for this pattern depicted in Figure 3 shows a payment service sending a payment to a retail service provider. The retail service provider sends a response indicating whether the payment details are valid or not. The interaction at the top shows the sending of the payment details by an actor playing the role "payment service" to an actor playing the role "retail service". These actors are referred to as "p1" and "s1" respectively. This interaction requires an acknowledgment (this is a design choice). The interaction at the bottom depicts the sending of the response. Again, this interaction requires acknowledgment. Both interactions are related via a precedes relationship, meaning that the lower interaction can only start if the upper interaction has been completed. In this case, the upper interaction is completed if the payment service has received the acknowledgment from the retail service. The local model for the actors participating in a given choreography would include every interaction where the party in question is participating. Thus, in the depicted example the local models for the participating parties "payment service" and "retail service" would be equivalent to the global model. Subsequently, we only show global models.

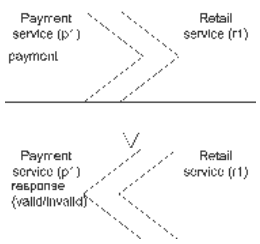

Fig. 3. Send/Receive (Pattern 3)

Racing incoming messages. In Figure 4 an example for the racing incoming messages pattern is depicted. This pattern is defined as follows [2]: "A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes." The figure depicts the scenario of a manufacturing process, which "involves remote subcontractors and uses a pull-strategy to streamline its operations. Each step in the manufacturing process is undertaken by a subcontractor. A subcontractor signals intention to execute a step when it becomes available through a request. At the same time, progress is monitored by a quality assurance service. The service randomly issues quality check requests in addition to the pre-established quality checkpoints in the process. When a quality check request arrives, it is processed in full before processing any new quality check request or subcontractor intention. Similarly, when a subcontractor intention arrives, it is processed in full before processing any other check request or subcontractor intention. Thus, there are points in the process where quality checks and subcontractor intentions compete." Figure 4 describes this point in the choreography. The two interactions at the top of the figure show the possible receipt of two different types of messages by the manufacturer: "manufacturing request" and "quality check request". These interactions are connected via a two-way inhibits relationship. In the figure an undirected crossed edge is used as abbreviation for two directed crossed edges. This indicates, that after one of the two messages has been received, the other one can no longer be received. With skipping one of these elementary interactions, the following elementary interaction will also be skipped, since the according prerequisite will never be fulfilled. The "manufacturing request" interaction precedes a "manufacturing approval" interaction while a "quality check request" interaction precedes a "quality check response" interaction.

One-to-many send/receive. The one-to-many send/receive pattern goes as follows [2]: "A party sends a request to several other parties, which may all be identical or logically related. Responses are expected within a given timeframe. However, some responses may not arrive within the timeframe and some parties may even not respond at all. The interaction may complete successfully or not depending on the set of responses gathered." The example for this pattern de-

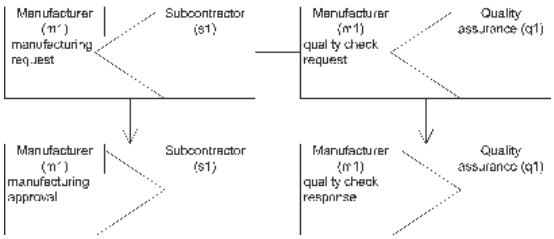

Fig. 4. Racing incoming messages (Pattern 4)

picted in Figure 5 shows a scenario, where "an insurance company outsources some aspects of its claims validation to its external search brokers. Brokers are typically small agencies and have variable demands. For efficiency, the insurance company sends search requests to all the brokers, and accepts the first three responses to undertake the search." The depicted interaction shows the repetition of the sending of the search requests by the insurance company to the search brokers and the according responses. The actors playing the roles of the insurance company and the search brokers are referred to as "i1" and "B1". As introduced in Section 3, "i1" is named actor reference. "B1" is denoting a set of actor references, indicated by starting with a capital letter instead of a small letter for a single actor reference. The rectangle surrounding the two interactions is depicting a repeated composite interaction, whereby the repetition instruction and an additional stop-condition for the number of concurrent executions are noted in small rectangles on top of the composite interaction. Both, the repetition instruction and the stop-condition are executed and evaluated respectively by the actor referred to as "i1" and playing the role "Insurance". The repetition instruction denotes, that the message has to be sent to all actors that are referred to in the set of actor references "B1". This set of actor references is bound by the actor executing the repetition instruction. All iterations are executed concurrent, which is noted in brackets after the repetition instruction. The whole repeated interaction is initializing a variable "responses", which is indicated by the content of the small rectangle below the repeated interaction. This variable is increased by the lower interaction, showing the receipt of the responses from the brokers. The value of this variable is part of the stop-condition, denoting the iteration is stopped if this variable has the value 3. The usage of this variable does not contradict the postulation of omitting variables, noted in the requirements section. This variable is necessary from a business point of view, since even responses that have been gathered need not necessarily change the value, e.g. if the response does not contain the requested information.

Multi-responses. The pattern multi-responses is defined in [2] as follows: "A party X sends a request to another party Y. Subsequently, X receives any number of responses from Y until no further responses are required. The trigger of no further responses can arise from a temporal condition or message content, and can arise from either X or Ys side. Responses are no longer expected from Y after

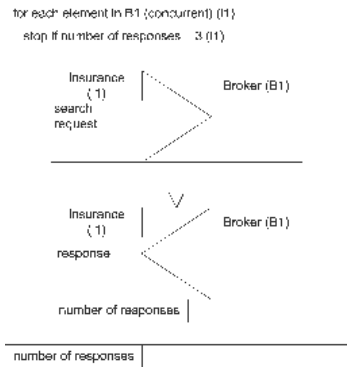

Fig. 5. One-to-many send/receive (Pattern 7)

one or a combination of the following events: (i) X sends a notification to stop; (ii) a relative or absolute deadline indicated by X; (iii) an interval of inactivity during which X does not receive any response from Y; (iv) a message from Y indicating to X that no further responses will follow. From this point on, no further messages from Y will be accepted by X." The example for this pattern depicted in Figure 6 shows a goods deliverer who is providing an urgent transportation service. "For optimization of travel, it subscribes to a local traffic reporting service provides its destination nodes (goods dispatch and customer locations) and obtains regular feeds on traffic bottlenecks, until it indicates that no feeds are required." The top left interaction depicts the subscription of the traffic service by the goods deliverer. After the completion of this interaction the two remaining interactions of the choreography are enabled. The interaction on the right-hand side shows the receipt of traffic information by the goods deliverer. This interaction is repeated sequentially until the iteration of the interaction is interrupted, since the stop condition will is always false. Accordingly, the stop-condition is evaluated by the actor referred to as "t1" and playing the role "Traffic service". The interruption of the repetition occurs, if the third interaction is completed, which shows the sending of an unsubscribe message from the goods deliverer to the traffic service. This interaction is thus connected via a inhibits relationship with the repeated interaction.

Relayed request. According to [2], the relayed request pattern is defined as follows: "Party A makes a request to party B which delegates the request to other parties (P1, ..., Pn). Parties P1, ..., Pn then continue interactions with party A while party B observes a view of the interactions including faults. The interacting parties are aware of this 'view' (as part of the condition to interact)." The example depicted for this pattern shows a government agency that outsources supportive work for managing regulatory provisions. Clients send requests to the government agency concerned by the regulation and the government agency forwards the request to the service providers. The government agency selects the service providers and the way they interact with the clients, e.g. key points of

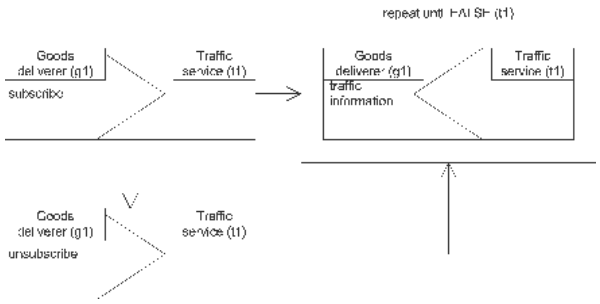

Fig. 6. Multi-responses (Pattern 8)

processing and key reports to be sent to the government agency. The above interaction shows the sending of a request from a client to the government agency. This interaction is connected via a precedes relationship with a repeated interaction, which is iterated concurrently for all service providers bound to a set of actor references "S1". The binding of this set of actor references, which is part of the repetition instruction, is executed by an actor referred to as "g1" and playing the role "Government". The government agency is delegating the request to each actor assigned to the set of actor references "S1" concurrently. This instruction for the number of executions is noted in a small rectangle on top of the repeated interaction. In this case there is no additional stop-condition for the "For each"-repetition, since it would equal to the maximum number of iterations expressed by the repetition instruction. During each iteration, the remaining two interactions are enabled after the service provider has received the request. The interaction on the right-hand side shows the sending of a response from the service provider to the client, while the interaction on the left side depicts the sending of a report to the government agency.

The solutions to the interaction patterns presented above and those given in appendix indicate that the Let's Dance language can deal with most relevant aspects necessary to model service choreographies. The following section presents a meta-model and an informal semantics of the language. In separate work [9] we have defined a formal semantics of the language by translation to π-calculus. We do not present details of this formalization here for space reasons.

5 Meta-model and informal semantics

Figure 8 provides an abstract syntax of the Let's Dance in the form of a meta-model captured in the Object-Role Modeling (ORM) notation [12]. The basic concept of the meta-model is that of a Communication Action, which is performed by an Actor (not shown in the diagram) designated by exactly one Actor Reference. An Actor Reference (or more specifically the actor it refers to) plays at least one Role and one Role is played by at least one Actor Reference. Communication Actions have exactly one type, which can be either Message Sending or

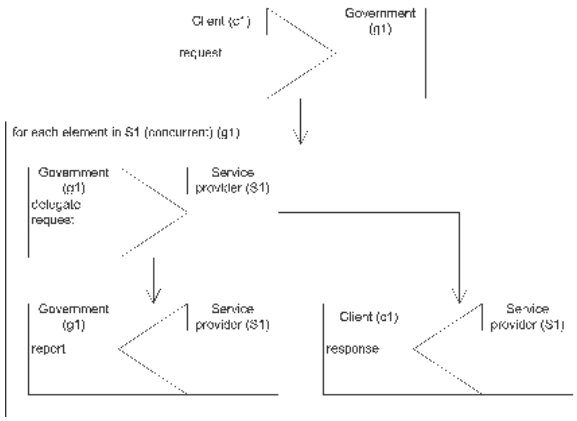

Fig. 7. Relayed Request (Pattern 12)

Message Receipt. These are the two basic communication primitives supported in service oriented architectures. Naturally, the party that performs the message sending is called the "sender" while the party that performs the message receipt is the "receiver". A Communication Action may require or may provide an acknowledgment. A Message Sending marked as "requiring acknowledgment" means that the sender expects to receive an "acknowledge" message from the receiver. In the case of a Message Receipt the receiver will send an "acknowledge" message after receiving a message if the Message Receipt is marked as "providing acknowledgment".

Communication Actions can write Variables and can be combined to form Interactions. Thereby a Message Sending action requiring acknowledgment (not requiring acknowledgment) can only be combined with a Message Receipt action providing acknowledgment (not providing acknowledgment).

An Interaction is a unit of information exchange and has exactly one type, which can be either Elementary (one-to-one) Interaction or Composite Interaction. If an Interaction is composed of other Interactions, we talk about the "sub-interactions" of a "super-interaction". Composite Interactions are a super-interaction of at least one Interaction and one Interaction can be the sub-interaction of at most one Composite Interaction. An Elementary Interaction involves two Communication Actions (one send and one receive) and corresponds to the logical exchange of a message from one party to another, that is, a party sends a message that the other party may receive. The sender can only perceive that the message was received if the interaction requires acknowledgment.

An Interaction has exactly one "Type of Repeated Interaction", which can be either None, Sequential for each, Concurrent for each, While or Until. Thus, the subtype Repeated Interaction has three specializations: For each, While and Until. The first one can refer to a Iteration Expression, which can be Actor-based (and thus consist of at least one Actor Reference) or Variable-based (and thus consist of at least one Variable). Additionally the Iteration Expression can have

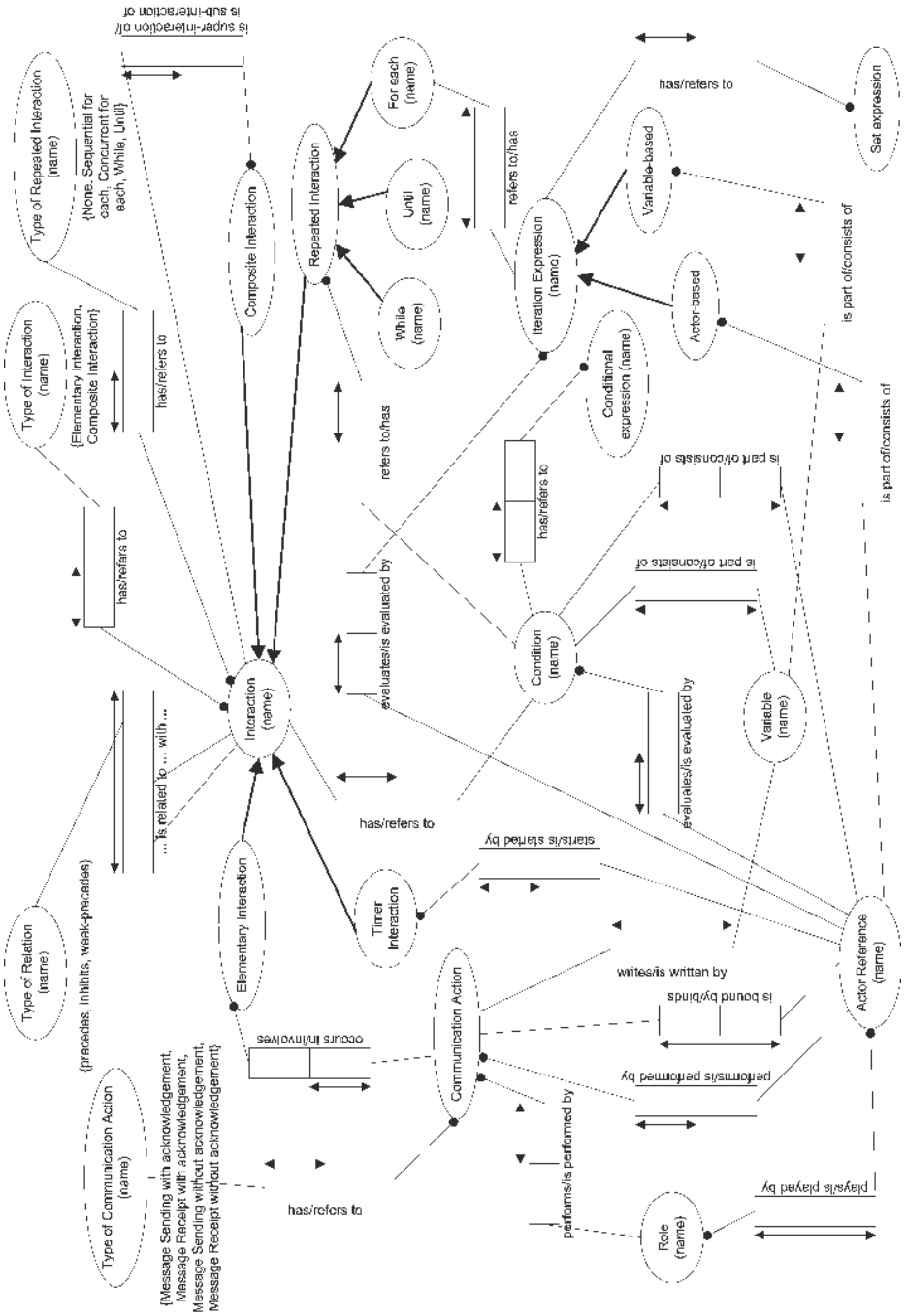

Fig. 8. Static Meta-model of the Language

at most one Set expression assigned, which allows for the elaboration of the repetition instruction. Let's Dance does not impose any particular expression language for describing set expressions and conditions. Arguably, domain analysts would want to specify these in natural language and let developers refine them into an executable expression language.

Every repeated interaction has exactly one (stop) condition assigned. If this condition evaluates to true, it implies that the iteration must stop (in case of "Until" and "For each") or must continue (in case of "While"). A condition has at most one Conditional expression assigned, which allows for the elaboration of the repetition instruction. Moreover Conditions can also be assigned to each type of Interaction, denoting a guard for the Interaction, whereby each Interaction has at most one guard. The associated Conditional expression allows for the elaboration of the guard instruction. A Condition is evaluated by at least one Actor Reference and can consist of multiple Variables and Actor References.

Actor References are bound during the execution of at least one Communication Action and in a given scenario each Communication Action can alter bindings executed earlier. This enables the passing of information where specific messages should be sent to, e.g. in a buyer-seller-shipper scenario, where the buyer is binding the actor reference of the shipper in order to nominate a transport service. If the nominated shipper is not available, the seller might alter the binding of the actor reference and choose another transport service. For Composite Interactions this leads to the possibility to change the recipient of messages during the execution of an Interaction. During execution, the information about the binding has to be sent with every message following the Interaction during which execution a specific Actor Reference has been bound, until the Communication Action is reached, which is executed by the bound Actor Reference.

Since a Condition is evaluated by at least one Actor Reference, it is possible to describe dependencies between Communication Actions and Interactions occurring earlier. Thus Conditions have to be evaluated before executing the respective Communication Action. For a condition evaluated by an actor sending a message, this means before initiating the send action and for a condition evaluated by an actor receiving a message before the completion of the receipt action (in other words before the consumption of the message), since in the latter case it may be necessary for the receiving party to be aware of the content of the message in order to evaluate a stated Condition. For an Actor Reference to be able to evaluate a Condition at all, the necessary information has to be available. This can only be ensured, if the information needed for the evaluation of a Condition is sent with every message following the Communication Action which initialized a certain Variable or Actor Reference, until the information reached the specified Communication Action and Interaction respectively that has the Condition assigned.

Any two interactions may be related by a "precedes", a "weak-precedes" or an "inhibits" relation. These relationships are defined as follows: an interaction X is said to precede another interaction Y, if Y can not occur before X has

been completed. If two interactions X and Y are connected with a weak-precedes relationship, then Y can not occur after X has been completed or after X has been skipped (i.e., seen from a global perspective, X will never occur). The inhibits relationship is defined as follows: an interaction X is said to inhibit another interaction Y, if Y can not occur any more after X has been completed. Moreover, the inhibits relationship is able not only to prevent the target interaction to be started, but also to interrupt it (and all of its sub-interactions), if it has already been started. These Relationships apply solely during the current execution of the least common super-interaction of the considered interactions. For example, if A and B are sub-interactions of interaction C and they are related through a precedes relationship, then during each execution of C, B can only occur after A has been completed.

The subtype Timer of an Interaction allows one to enforce time limits on other interactions. It is started by exactly one actor (reference). A Timer is defined as a composition of two Elementary Interactions with a fixed party "Clock". The first Elementary Interaction is the arming of the Timer by sending a Message to the Clock (and the Clock receiving the Message), while the second Elementary Interaction is the sending of a Message from the Clock in order to indicate that the specified time period has expired. The latter of these messages is only sent if a condition is evaluated to true, which compares the elapsed time since the arming of the Timer with the period specified in the first Message. Since a Timer is a subtype of Interaction it can be related to other Interactions arbitrarily.

6 Conclusion and future research directions

In this paper, we have motivated the need for a service behavior modeling language, spelled out a number of associated requirements, and proposed an initial version of a language (Let's Dance) that fulfills the most crucial of these requirements, namely abstraction, comprehensibility, and suitability. Unlike existing service behavior description languages such as BPEL and WS-CDL, which focus on supporting the implementation phase, the proposed language is not based on imperative programming constructs such as variable assignment, if-then-else and switch statements, sequence, and while loops. Also, the language supports the description of both local and global views of service interactions (i.e. behavioral interfaces and choreographies respectively).

The suitability of the language has been demonstrated on the basis of scenarios corresponding to 13 patterns of service interaction previously identified. The paper also presented an abstract syntax of the language in the form of a static meta-model, as well as an informal semantics. For a formal execution semantics of Let's Dance, defined in terms of a translation to π-calculus, we refer to [9]. This work also discusses the issue of reachability analysis of Let's Dance choreographies (i.e. detecting interactions in a choreography that will never be executed). A more in-depth discussion on desirable properties of Let's Dance choreographies is provided in [20]. In particular, this latter reference discusses the issue of local enforceability of Let's Dance choreographies, which is a pre-

requisite to generating local models from choreographies. It turns out that not all choreographies defined as flows of interactions (the paradigm adopted in Let's Dance) can be mapped into local models that satisfy the following conditions: (i) the local models contain only interactions described in the choreography; and (ii) they collectively enforce all the constraints in the choreography. Proposals around WS-CDL skirt this issue. Instead, they assume the existence of a state (i.e. a set of variables) shared by all participants. Participants synchronize with one another to maintain the shared state up-to-date. Thus, certain interactions take place between services for the sole purpose of synchronizing their local view on the shared state and these interactions are not defined in the choreography. In the worst case, this leads to situations where a business analyst signs off on a choreography, and later it turns out that to execute this choreography a service provided by one organization must interact with a service provided by a competitor, unknowingly of the analyst. Thus, it is desirable to provide tool support to analyze choreographies to determine whether or not they are enforceable by some set of local models.

In [8], we present a tool that implements algorithms for static analysis of Let's Dance choreographies (including reachability and enforceability analysis) and for generation of local models. Ongoing work is concentrating on defining a translation from Let's Dance local models into BPEL code.

Acknowledgment The first author is funded in part by SAP. The third author is funded by a "Smart State" Fellowship co-sponsored by Queensland Government and SAP.

References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services, version 1.1*, May 2003. Available at: `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`
2. A. Barros, M. Dumas, and A. H.M. ter Hofstede. Service Interactions Patterns In *Proceedings of the 3rd International Conference on Business Process Management (BPM)*, Nancy, France, September 2005. Springer Verlag, pp. 302-218. Extended version available as Technical Report FIT-TR-2005-02, Faculty of IT, Queensland University of Technology, `http://eprints.qut.edu.au/archive/00002295`
3. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. *Web Services Choreography Description Language Version 1.0*, W3C Candidate Recommendation, November 2005. `http://www.w3.org/TR/ws-cdl-10`.
4. B. Benatallah, F. Casati, F. Toumani, and R. Hamadi. Conceptual Modelling of Web Service Conversations. In *Proceedings of 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Velden, Austria, June 2003. Springer Verlag, pp. 449-467.
5. A. F. Blackwell, K. N. Whitley, J. Good, M. Petre. Cognitive Factors in Programming with Diagrams. *In Artificial Intelligence Review* 15(1/2): 95-114 (2001)
6. J. Clark, C. Casanave, K. Kanaskie, B. Harvey, J. Clark, N. Smith, J. Yunker, K. Riemer (Eds). *ebXML Business Process Specification Schema Version 1.01*,

UN/CEFACT and OASIS Specification, May 2001. http://www.ebxml.org/specs/ebBPSS.pdf.
7. W. Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1), pages 45–80, Hingham, MA, USA, 2001. Kluwer Academic Publishers.
8. G. Decker, M. Kirov, J. M. Zaha, M. Dumas. Maestro for Let's Dance: An Environment for Modeling Service Interactions. In *Demonstration Session of the 4th International Conference on Business Process Management (BPM)*, Vienna, Austria, September 2006.
9. G. Decker, J. M. Zaha, M. Dumas. Execution Semantics for Service Choreographies. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM)*, Vienna, Austria, September 2006. Springer Verlag.
10. H. Foster, S. Uchitel, J. Magee, J. Kramer. Tool Support for Model-Based Engineering of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Servies (ICWS)*, Orlando FL, USA, July 2005. IEEE Computer Society.
11. T. R. G. Green, M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing* 7(2):131-174, 1996.
12. T. Halpin. *Information Modeling and Relational Databases - From onceptual Analysis to Logical Design.* Morgan Kaufman, 2001.
13. K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use – Volume 1.* Springer-Verlag, Berlin, 1997.
14. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, July 2004. Springer, pp. 26–42.
15. Object Mangement Group (OMG): *UML Profile for EDOC*. February 2004. http://www.omg.org/technology/documents/formal/edoc.htm
16. Object Management Group (OMG): *UML 2.0 Superstructure Specification*. OMG Document ptc/04-10-02, October 2004. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02
17. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology* 1(1):77–106, 2005.
18. RosettaNet: Partner Interface Protocols. http://www.rosettanet.org
19. S. White: *Business Process Modeling Notation (BPMN)* – Version 1.0, May 2004, http://www.bpmi.org.
20. J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, G. Decker. Service Interaction Modeling: Bridging Global and Local Views. In *Proceedings of the Tenth IEEE International Conference on Enterprise Distributed Object Computing (EDOC)*, Hong Kong, China, October 2006. IEEE Computer Society.

A Mapping of Patterns

In the following the 13 interaction patterns from [2] are modeled with the proposed language. For each of the patterns the examples given in [2] are included. Each example is modeled with the proposed language and an explanation of the respective figure in natural language is given.

A.1 Single-transmission bilateral interaction patterns

Pattern 1: Send

- Description: A party sends a message to another party.
- Example: An alerting service sends a reminder of an anniversary to a registered user.

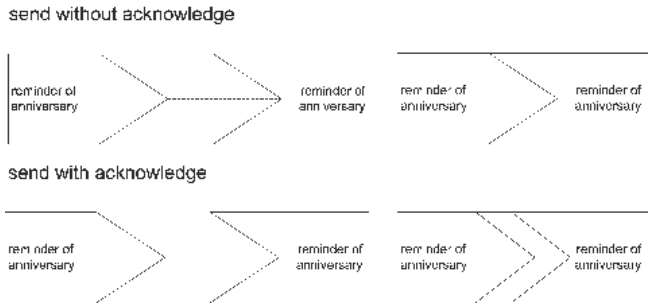

Fig. 9. Pattern 1: Send

- Diagram (Fig. 9): The depicted diagram shows the two different types of sending a message. The above part of the figure shows the sending without acknowledgement. On the left-hand side the connected communication actions are depicted, while on the right-hand side the juxtaposition of the symbols forming an elementary interaction is depicted. The lower part of the figure shows the constructs for the sending of a message with acknowledge. Thereby the acknowledgement message is not depicted. The abbreviation depicted asides is a rectangle, which has two arrows in the middle.

Pattern 2: Receive

- Description: A party receives a message from another party.
- Example: A purchasing service receives a notification of delivery delay from a shipping service.
- Diagram (Fig. 10): Compared with the figure for sending a message, the receipt of a message is just a mirror-inverted depiction of the modeling constructs.

Pattern 3: Send/receive

- Description: A party X engages in two causally related interactions: in the first interaction X sends a message to another party Y (the request), while in the second one X receives a message from Y (the response).
- Example: A payment service sends a payment to a retail service provider who either sends back a message indicating that the payment details are invalid, or a receipt.

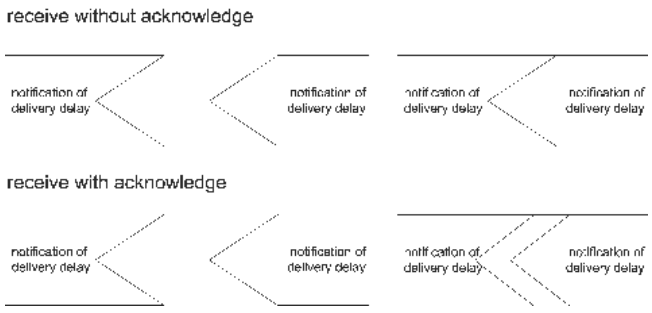

Fig. 10. Pattern 2: Receive

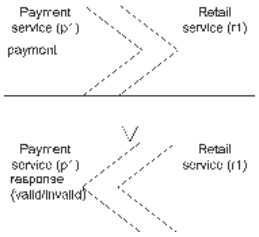

Fig. 11. Pattern 3: Send/receive

– Diagram (Fig. 11): Both messages, the sending of the payment as well as the response message, which content is determining weather the payment details were valid or not, are sent with acknowledgement. This is indicated by the respective symbols. For denoting that a communication action is executed by a specific role, the name of this role is noted starting with a capital letter inside a small rectangle at the top of the respective symbol for the communication action. The actor reference playing this role is noted in brackets after the nomination of the role. In the depicted example "p1" is playing the role of a "Payment service" and "r1" is playing the role of a "Retail service". The directed edge between the two interactions symbolizes the precedes relationship between them, since the payment service has to receive the message in order to decide weather the payment details are valid or not.

A.2 Single-transmission multilateral interaction patterns

Pattern 4: Racing incoming messages

– Description: A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes.

– Example 1: A manufacturing process involves remote subcontractors and uses a pull-strategy to streamline its operations. Each step in the manufacturing process is undertaken by a subcontractor. A subcontractor signals intention to execute a step when it becomes available through a request. At the same time, progress is monitored by a quality assurance service. The service randomly issues quality check requests in addition to the pre-established quality checkpoints in the process. When a quality check request arrives, it is processed in full before processing any new quality check request or subcontractor intention. Similarly, when a subcontractor intention arrives, it is processed in full before processing any other check request or subcontractor intention. Thus, there are points in the process where quality checks and subcontractor intentions compete.

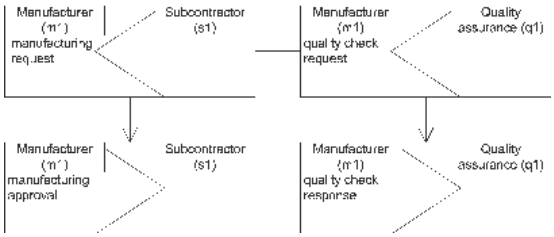

Fig. 12. Pattern 4: Racing incoming messages - Example 1

– Diagram 1 (Fig. 12): The above two interactions show the receipt of two different types of messages by the manufacturer: “manufacturing request” and “quality check request”. These interactions are connected via a two-way inhibits relationship. In the figure an undirected crossed edge is used as abbreviation for two directed crossed edges. This indicates, that after one of the two messages has been received, the other one can no longer be received. With skipping one of these elementary interactions, the following elementary interaction will also be skipped, since the according prerequisite will never be fulfilled. The “manufacturing request” interaction precedes a “manufacturing approval” interaction while a “quality check request” interaction precedes a “quality check response” interaction.
– Example 2: The escalation service of an insurance company’s call center may receive storm alerts from a weather monitoring service (which typically herald surges in demand), notifications of long waiting times from the queue management service, or notifications of low resourcing levels from the call center’s HR manager. The receipt of any of these three types of messages by the escalation service triggers an escalation process (different processes apply to the various types of notifications). While an escalation process is running, subsequent storm alerts, queue saturation or low resourcing notifications are made available to the call center manager but will not trigger new escalations.

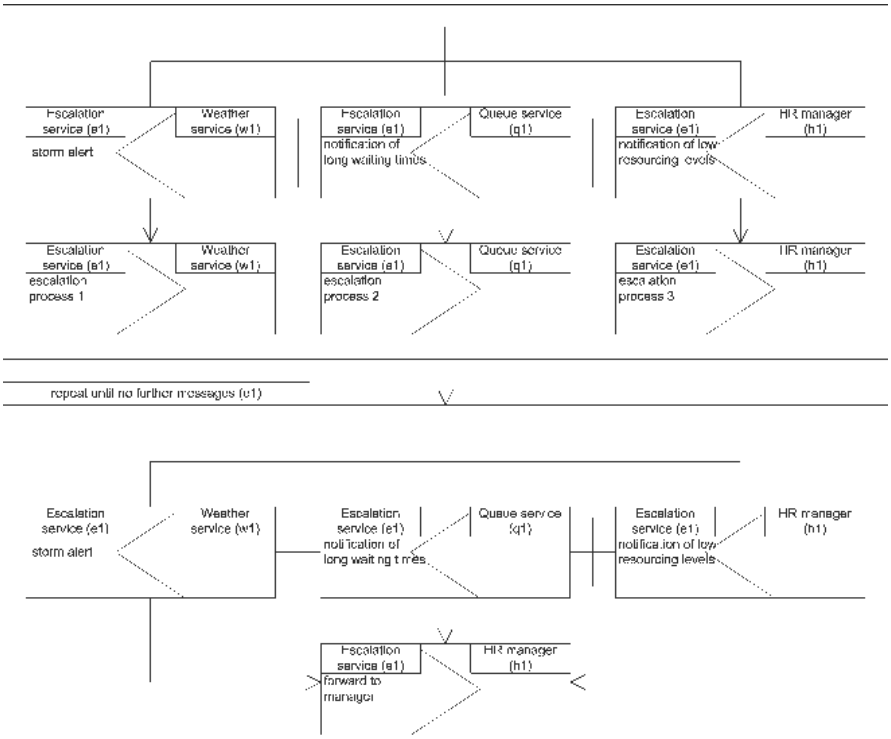

Fig. 13. Pattern 4: Racing incoming messages - Example 2

– Diagram 2 (Fig. 13): The whole choreography for racing incoming messages consists of two composite interactions. The composite interaction depicted above shows the three types of possible incoming messages which are all received by an actor referred to as "e1" and playing the role of the escalation service. The messages are received from actors playing the role "Weather service" ("storm alert"), "Queue service" ("notification of long waiting times") and "HR manager" ("notification of low resourcing levels"). Each of these interactions is connected via a precedes relationship with the messages for triggering the corresponding escalation processes. Moreover each of them is connected via a inhibits relationship with the remaining two of these three interactions, denoting that after the receipt of one of the messages the other messages can not be received anymore. This can either be indicated by a crossed directed edge or, like it is depicted in the example, by using the abbreviation for reciprocally connected interactions. The composite interaction is connected with a precedes relationship with another composite interaction, which can be repeated until no further messages arrive. This is denoted by the content of the small rectangle at the top of the rectangle, containing the interaction to be repeated. The stop-condition is evaluated by the actor

referred to as “e1” and playing the role of the escalation service. At top of this interaction the constellation for the three possible incoming messages is depicted like in the above composite interaction. Each of the three interactions is connected with an interaction that denotes the forwarding of the received message to the HR manager.

Pattern 5: One-to-many send

- Description: A party sends messages to several parties. The messages all have the same type (although their contents may be different).
- Example 1: A purchasing service sends a call for tender to all known trading parties that provide a given type of product or service.

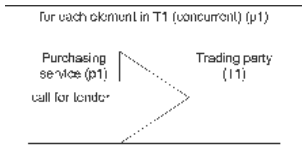

Fig. 14. Pattern 5: One-to-many send - Example 1

- Diagram 1 (Fig. 14): The diagram shows a composite interaction that consists of one elementary interaction being repeated for a set of trading partners “T1”, which is denoted by the content of the small rectangle on top of the repeated interaction. Therefore the actor referred to as “p1” and playing the role “Purchasing service” has to execute the repetition instruction and therewith to bind the set of actor references “T1”. The according stop-condition is false and is thus omitted in the depicted example. The fact of “T1” being a set of actor references and not a single actor reference is indicated by noting the name with a starting capital letter. The purchasing service is sending a call for tender to these trading partners. The communication action constituting the actual receipt of the tender is executed by the actors bound to the set of actor references “T1”.
- Example 2: An accreditation authority sends a notification to all registered candidates who have passed a certification test. Each of these notifications contains information that is specific to the recipient (e.g. test results).
- Diagram 2 (Fig. 15): Like in the first example for this pattern, there is one composite interaction repeating an elementary interaction. Here, the repeated interaction denotes the message sending to a set of candidates, which is referred to as “C1”. The repetition is executed concurrent for this set of candidates, whereby this set of actor references has to be bound by the actor referred to as “a1” and playing the role “Accreditation”.

Pattern 6: One-from-many receive

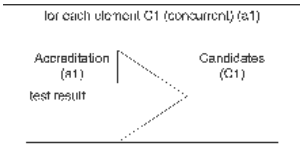

Fig. 15. Pattern 5: One-to-many send - Example 2

– Description: A party receives a number of logically related messages that arise from autonomous events occurring at different parties. The arrival of messages needs to be timely enough for their correlation as a single logical request. The interaction may complete successfully or not depending on the set of messages gathered.
– Example 1: Multi-part order. A printer operating in a high-demand market allows a streamlining of job preparation whereby documents belonging to the same job can be supplied directly by different parties (as with legislative documents by governments requiring quick turnaround). The first notification carries the job requirements including the delivery time and location. This and subsequent messages provide document content. Several interactions take place between the printer and the primary partner to mark-off the production steps, e.g. typesetting preview and payment. The printer's policy has a deadline for collection of all source documents in order to meet production deadlines. If the sources are not available by this deadline, the interaction fails, otherwise it succeeds.

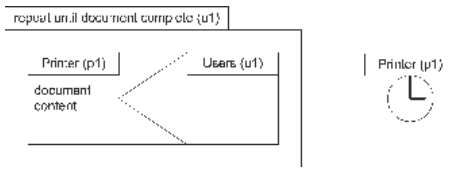

Fig. 16. Pattern 6: One-from-many receive - Example 1

– Diagram 1 (Fig. 16): The receipt of multiple messages that contain document content is depicted as a repeated interaction, which is iterated until the document is complete. The stop-condition for the concurrent iteration is evaluated by the actor referred to as "u1" and playing the role "User". The different parts of the document have to be gathered within a certain timeframe, which is defined by the printer starting a timer. The repeated interaction and the timer are connected with a two-way inhibits relationship. Thus, in case of the timer expiring before all document parts can be gathered, the repeated interaction is interrupted. In case of all document parts being

gathered before the timer expires, the repeated interaction is completed and the timer is interrupted.

– Example 2: Batched requests. A group buying service receives requests for buying different types of items. When a request for buying a given type of product is received, and if there are no other pending requests for this type of item, the service waits for other requests for the same type of item. If at least three requests have been received within five days, a "group request" is created and an order handling process is started. If on the other hand less than three requests are received within the five days timeframe, the requests are discarded and a fault notification is sent back to the corresponding requestors.

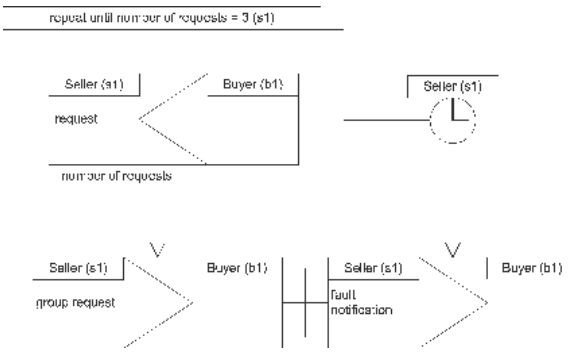

Fig. 17. Pattern 6: One-from-many receive - Example 2

– Diagram 2 (Fig. 17): In the figure the above composite interaction depicts the incoming requests for buying a given product. Each incoming request increases the variable "number of request", which is initialized by starting the composite interaction and and tested within the condition of the composite interaction, i.e. before each receipt of a request. The timer with the five days deadline is started by the actor referred to as "s1" and playing the role "Seller". The same actor is evaluating the stop-condition for the repetition. If three requests are received within five days, the below interaction on the left side is enabled and the timer is interrupted. The enabled interaction is depicting the sending of the group request and is disabling the interaction for sending a failure notification. If the timer expires before three requests have been received, the repeated interaction for receiving requests is interrupted and the interaction for sending a failure notification is enabled. For connecting the repeated interaction and the timer and the two interactions at the bottom respectively, a two-way inhibits relationship is used.
– Example 3: Event filtering prior to persistence. Investment consultants subscribe to a stock market watch service which broadcasts significant trading events. This allows the consultants to provide timely recommendations to their customers for changes in share portfolios. Events are correlated into

composite events per fund manager. Because investment consultants are typically small enterprises (or smaller units of large enterprises), they cannot afford large databases in order to persist the high-volumes of incoming events prior to correlation. Rather they apply correlation rules as the events become available. If the individual events successfully correlate within the relevant timeframe into a significant composite event, an escalation is triggered. Otherwise the events are ignored.

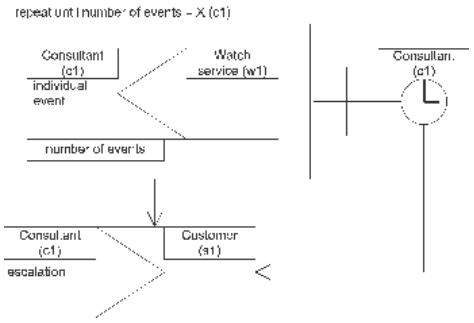

Fig. 18. Pattern 6: One-from-many receive - Example 3

– Diagram 3 (Fig. 18): The only difference in this example compared to the second example of this pattern is the missing of an interaction that is executed in case of the timer expiring before the composite interaction can be completed. Thus, the timer can be connected via a inhibits relationship with the elementary interaction below, showing the sending of a group event after "X" individual events have been received.

Pattern 7: One-to-many send/receive

– Description: A party sends a request to several other parties, which may all be identical or logically related. Responses are expected within a given timeframe. However, some responses may not arrive within the timeframe and some parties may even not respond at all. The interaction may complete successfully or not depending on the set of responses gathered.
– Example 1: RosettaNet Partner Interface Process (PIP) 3A3 "Request Price and Availability" (http://www.rosettanet.org/PIP3A2). In this process, a buyer identifies a number of potential suppliers and sends a request for "price and availability" to each of them. Responses from all of these suppliers are then gathered and analysed. The number of potential suppliers is not known at design time.
– Diagram 1 (Fig. 19): The whole choreography consists of a repeated interaction, that is repeated for each element in a set of actor references "S1". The repetition instruction is executed by the actor referred to as "b1" and playing the role "Buyer". The stop-condition is omitted in the depicted example,

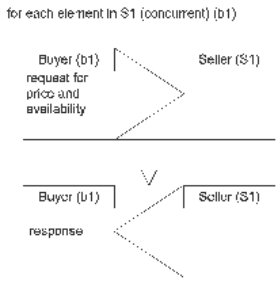

Fig. 19. Pattern 7: One-to-many send/receive - Example 1

because the request should be sent to all elements in "S1". The repeated interaction has two elementary sub-interactions that are connected with a precedes relationship. The Interaction noted above shows the sending of a request for price and availability to a set of suppliers. The lower interaction depicts the receipt of the responses by the buyer.

– Example 2: An insurance company outsources some aspects of its claims validation to its external search brokers. Brokers are typically small agencies and have variable demands. For efficiency, the insurance company sends search requests to all the brokers, and accepts the first three responses to undertake the search.

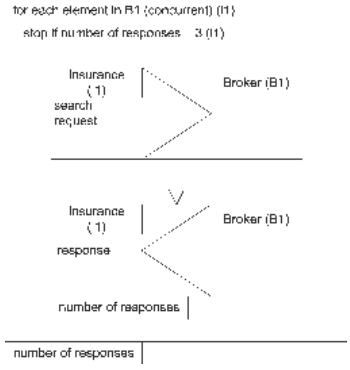

Fig. 20. Pattern 7: One-to-many send/receive - Example 2

– Diagram 2 (Fig. 20): The depicted interaction shows the repetition of the sending of the search requests by the insurance company to the search brokers and the according responses. The actors playing the roles of the insurance company and the search brokers are referred to as "i1" and "B1". The rectangle surrounding the interaction is depicting the repetition of the interaction, whereby the repetition instruction and the condition for the number

of concurrent executions are noted in small rectangles on top of the repeated interaction. Both, the repetition instruction and the stop-condition are executed and evaluated respectively by the actor referred to as "i1" and playing the role "Insurance". The repetition instruction denotes, that the message has to be sent to all actors that are part of the set of actor references "B1". This set of actor references is bound by the actor executing the repetition instruction. All iterations are executed concurrent, which is noted in brackets after the repetition instruction. The whole repeated interaction is initializing a variable "responses", which is indicated by the content of the small rectangle below the repeated interaction. This variable is increased by the lower interaction, showing the receipt of the responses from the brokers. The value of this variable is part of the stop-condition, denoting the iteration is stopped if this variable has the value 3.

A.3 Multi-transmission interaction patterns

Pattern 8: Multi-responses

- Description: A party X sends a request to another party Y. Subsequently, X receives any number of responses from Y until no further responses are required. The trigger of no further responses can arise from a temporal condition or message content, and can arise from either X or Y's side. Responses are no longer expected from Y after one or a combination of the following events: (i) X sends a notification to stop; (ii) a relative or absolute deadline indicated by X; (iii) an interval of inactivity during which X does not receive any response from Y; (iv) a message from Y indicating to X that no further responses will follow. From this point on, no further messages from Y will be accepted by X.
- Example 1: Order forecasting. As part of order forecasting, a buyer sends a request to a seller providing point-of-sale data, events impacting on order forecast, e.g. new products, stores opening/close, inventory strategy data and current inventory position (on hand, in transit, on order). The seller provides response data as various investigations make that data available. The investigations relate to subcontractors in manufacturing and delivery, market studies (historical demand data), and logistical delivery data (capacity related, lead times, delivery operations schedules).
- Diagram 1 (Fig. 21): The upper interaction on the left-hand side shows the sending of a request from the buyer to a seller referred to as "s1". This interaction is connected via a precedes relationship with a repeated interaction, that shows the sending of the request for data to a set of subcontractors by the seller and the subsequent gathering of necessary data for performing the forecast. Therefore the set of actor references "C1", which refers to the subcontractors, has to be bound while executing the repetition instruction. This is performed by the actor referred to as "s1" and playing the role "'Seller". The same actor is evaluating the stop-condition, which is stated in the second rectangle on top of the repeated interaction. If the stop-condition

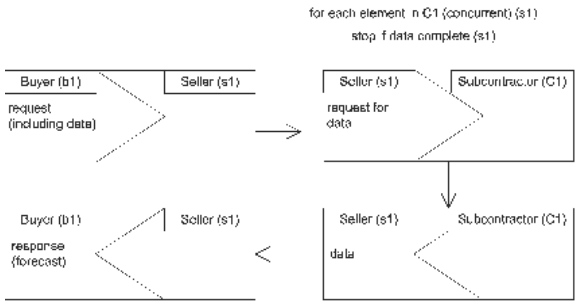

Fig. 21. Pattern 8: Multi-responses - Example 1

has been fulfilled, the seller is sending the final response to the buyer that initiated the process.

– Example 2: News refresh. A goods deliverer provides an urgent transportation service on behalf of suppliers to customers in a city. For optimization of travel, it subscribes to a local traffic reporting service provides its destination nodes (goods dispatch and customer locations) and obtains regular feeds on traffic bottlenecks, until it indicates that no feeds are required.

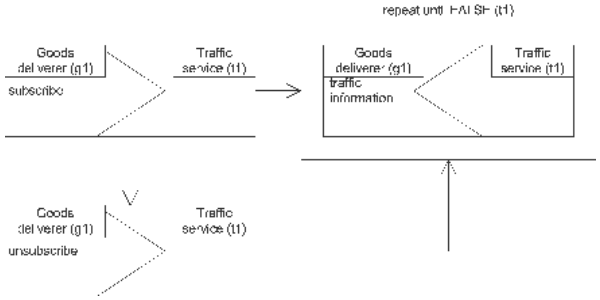

Fig. 22. Pattern 8: Multi-responses - Example 2

– Diagram 2 (Fig. 22): The interaction on top of the left-hand side shows the subscription of the traffic service by the goods deliverer and is connected via a precedes relationship with the lower elementary interaction. This interaction depicts the unsubscription of the traffic service and is connected with a inhibits relationship with the repeated interaction, which is denoting the receipt of traffic information by the good deliverer. If the goods deliverer is sending a “unsubscribe” message and this message has been received by the traffic service, the repeated interaction is interrupted and the choreography is completed.

Pattern 9: Contingent requests

– Description: A party X makes a request to another party Y. If X does not receive a response within a certain timeframe, X alternatively sends a request to another party Z, and so on.
– Example 1: An online service supports document submissions to academic conferences, tenders and grant/funding schemes. Applicants span different geographic boundaries for the different applications. Large bottlenecks are experienced near deadlines of especially high-volume applications. Proposals can be directly uploaded wherein further interactions are required to entry and validation of proposal details. Alternatively, different servers are available in different geographic localities for queue/forward of proposals to the central repository. The validation steps of proposals take place once the proposal has been forwarded (a notification is sent to the applicant to trigger this). In first instance, the system attempts to submit the proposal for direct upload to the central repository, but if this service is unavailable or overloaded, it will try the "nearest" queuing service, and so on with other queuing services.

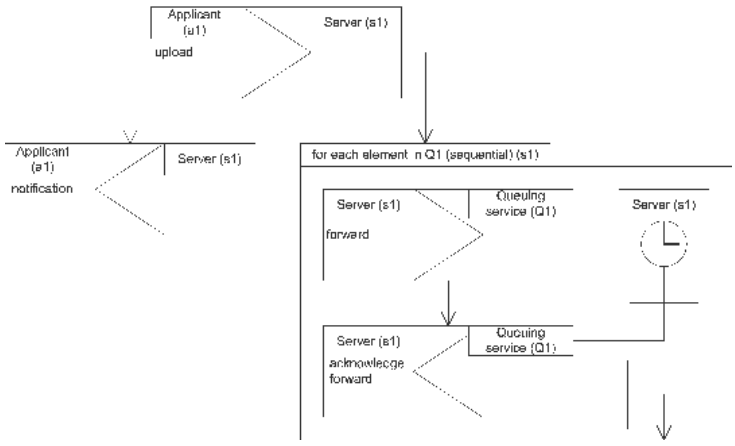

Fig. 23. Pattern 9: Contingent requests - Example 1

– Diagram 1 (Fig. 23): The elementary interaction on top shows the upload of the document to the server by the applicant. This interaction enables a repeated interaction and the notification of the applicant, indicating that the document was validated. The repeated interaction denotes the sending of the document to a set of queuing services (including the central repository) and the receipt of an acknowledgement. The set of queuing services "Q1" is bound as part of the repetition instruction, whereby the according stop-condition is omitted. The repetition instruction is executed by an actor referred to as "s1" and playing the role "Server". Each iteration of the composite interaction starts with the forward of the proposal and the start of the timer by the server, indicating how long the server will wait for the acknowledgement

that the document has been forwarded. If the acknowledgement has been received within the specified timeframe, the timer is interrupted as well as the whole repeated interaction. Otherwise the timer interrupts the waiting for acknowledgement and the forwarding is repeated.
– Example 2: A travel agency allows contingent reservations of flights in particular situations - urgent requests and busy flight paths. Customers nominate the preference of flight carriers. In order of preference, reservations are sought in short-timeframes. If a reservation is secured, the interaction ends.

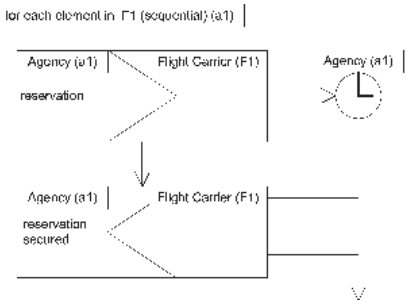

Fig. 24. Pattern 9: Contingent requests - Example 2

– Diagram 2 (Fig. 24): The depicted example is similar to the previous one. The main difference is the missing crossing of the repeated interaction. The repeated interaction of the diagram shows the reservation requests that are sent to each one of the nominated set of flight carriers. After the timer has expired, one iteration is finished and the request is sent to the next flight carrier. Each of the reservations sent is a prerequisite for a reservation to be secured and thus the interaction for sending a reservation is connected with a precedes relationship with the interaction for receiving a notification that a reservation has been secured. If this interaction is completed, the repetition is interrupted and the choreography ended.

Pattern 10: Atomic multicast notification

– Description: A party sends notifications to several parties such that a certain number of parties are required to accept the notification within a certain timeframe. For example, all parties or just one party are required to accept the notification. In general, the constraint for successful notification applies over a range between a minimum and maximum number.
– Example 1: Classical atomicity. A business venture service supports the process of business license applications for various small business endeavors (e.g. opening a restaurant). After the steps of obtaining and verifying application details, relevant agencies involved in the approval or registration of the application are notified. All of them must receive notification as there are inter-dependent aspects of the application leading to crossconsultation.

There may also be competing applications for the same business (down to the same location). Therefore, all agencies should receive the notification in a timely fashion. In this example, the minimum and maximum equal the number of all agencies notified.

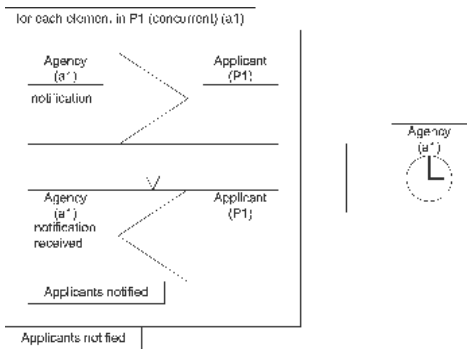

Fig. 25. Pattern 10: Atomic multicast notification - Example 1

- Diagram 1 (Fig. 25): The depicted example consists of a timer and a repeated composite interaction. The repeated interaction is initializing a variable "Applicants notified" and consists of two elementary interactions that are connected with a precedes relationship. The upper elementary interaction depicts the sending of the notifications to a set of applicants. Therefore the actor referred to as "a1" and playing the role "Agency" has to bind the set of actor references "P1", because this actor is the one executing the repetition instruction. The lower interaction shows the receipt of the acknowledgement and increases the variable "Applicants notified". In this example the variable is not used, since the number of applicants that have to be notified equals to all elements of "P1". If this number would differ, the variable would be used in a stop-condition that had to be added. If the timer expires before all acknowledgements have been gathered, the timer interrupts the repeated composite interaction. If all acknowledgements are gathered within the specified timeframe, the timer is interrupted.
- Example 2: Competing through atomicity. A legal firm has automated its property conveyance process for various loan types. The process utilizes a number of search brokers who have the same level of service agreements with the firm (service level agreements, e.g. cots model and timeliness of response obligations). Each of the brokers competes for conveyance applications. Therefore, only one of the notified brokers is selected, namely the first to accept the request. The minimum and maximum both are one.
- Diagram 2 (Fig. 26): In this example the repeated composite interaction can not be interrupted by a timer. Thus, the initiating party will wait until the first accept request arrives, no matter how long this takes. The sending of the request to all elements of a set of search brokers "S1" is executed by a

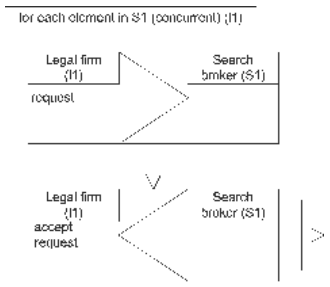

Fig. 26. Pattern 10: Atomic multicast notification - Example 2

actor referred to as "l1" and playing the role "Legal firm". As soon as the first accept has been received, the repeated interaction is inhibited and the choreography is ended.

– Example 3: Nesting. A travel agent allows the booking of both international and domestic travel requirements as part of comprehensive travel packaging. Customers nominate their preferred flight carriers as well as domestic requirements such as hotel accommodation, local travel bookings/hire and reservations to key attractions. In this example, each node in the international flight path can be seen as atomic group, and within a group, the flight carrier, booking agencies for local hotels, travel, care hire and so on, identify the services contacted. The different groups can have different minimum and maximum constraints depending on the domestic requirements of the customer. However, all atomic groups need to succeed in order for the interaction as a whole to succeed. (It should be noted that an extension of this example could see further levels of nesting within the atomic groups).

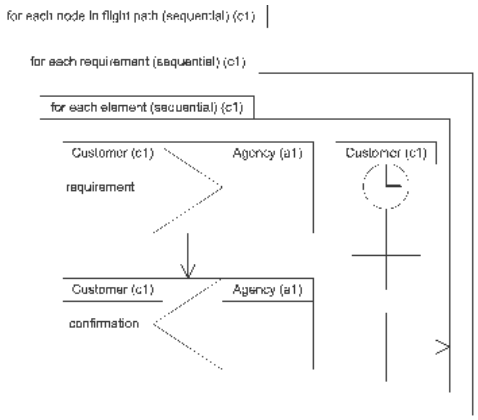

Fig. 27. Pattern 10: Atomic multicast notification - Example 3

– Diagram 3 (Fig. 27): The whole choreography consists of three nested repeated interaction, whereby the inner repeated interaction consists of two elementary sub-interactions that are connected with a precedes relationship. The outer interaction denotes the repetition for each node in the flight path, executed by an actor referred to as "c1" and playing the role customer. For each node, the repetition of the middle repeated interaction is started, which denotes the different requirements like accommodation, local travel booking or car. Finally, for each of these requirements, the inner repetition iterates over the preferred providers of these requirements. For each of the possible providers, the customer sends a message to the agency and starts a timer. If the timer expires before a confirmation is received, the timer inhibits the confirmation and the customer is able to send a new message to the agency. As soon as the customer receives the first confirmation of a possible provider, the timer and the inner repeated interaction are interrupted and the next requirement is considered.

A.4 Routing patterns

Pattern 11: Request with referral

– Description: Party A sends a request to party B indicating that any follow-up response should be sent to a number of other parties (P1, P2, , Pn) depending on the evaluation of certain conditions. While faults are sent by default to these parties, they could alternatively be sent to another nominated party (which may be party A).
– Example 1: Referral to single party: As part of a purchase order processing, a supplier sends a shipment request to a transport service. Subsequently, the transport service reports shipment status (e.g. as per RosettaNet's PIP 3B1) directly to the customer who then correlates these with its initial purchase order.

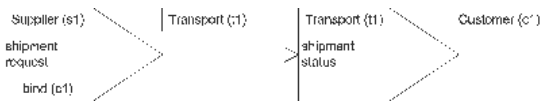

Fig. 28. Pattern 11: Request with referral - Example 1

– Diagram 1 (Fig. 28): The interaction depicted on the left side of the figure shows the sending of a shipment request from a supplier to a transport service. The actor referred to as "s1" and playing the role "Supplier" is binding the actor reference "c1", indicated by the content of the small rectangle at the bottom of the respective communication action. This interaction is connected with a precedes relationship with the interaction on the right-hand side, depicting the communication between the transport service and the customer. For indicating, that the bound actor reference should execute the

communication action for receiving the shipment status, the actor reference is defined as playing the role "Customer" during this communication action. Since the supplier bound this actor reference, the supplier is able to define where the shipment status should be sent to.

– Example 2: Referral to multiple parties: After processing its inventory restocking for a week, a supermarket's warehouse contacts a supplier for order and dispatch of goods, notifying it of the different transport services available (different services specialize in transport of different sorts of goods). The supplier directly interacts with these transport services regarding the scheduled dispatch times (arranged by the supermarket). Faults related to order fulfillment are sent by the supplier to the warehouse, while faults related to delivery are sent by the corresponding transport services to the warehouse.

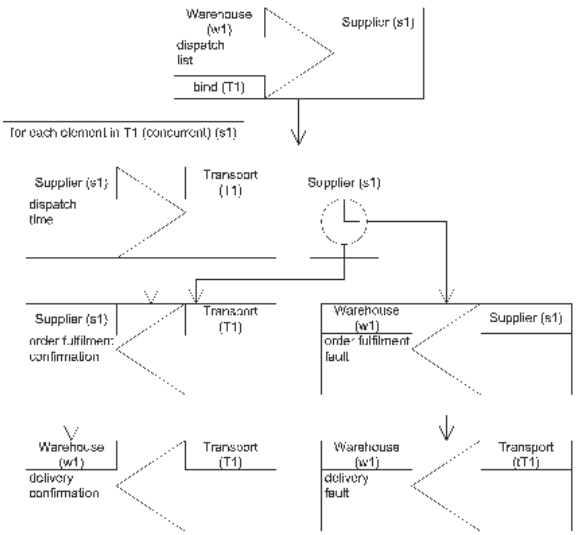

Fig. 29. Pattern 11: Request with referral - Example 2

– Diagram 2 (Fig. 29): The above elementary interaction shows the sending of the information about available transport services from the warehouse to the supplier. The actor referred to as "w'" and playing the role "Warehouse" is binding a set of actor references "T1", for which elements the enabled composite interaction noted below is repeated. In this composite interaction "T1" is playing the role "Transport". The supplier is sending the dispatch times to each of the transport services that have been noted by the warehouse. Additionally the supplier is starting a timer. The sending of the dispatch times enables the receipt of a confirmation of the order fulfilment by the supplier. If this confirmation is not received within a certain timeframe, this interaction is interrupted and the interaction, which informs

the warehouse about an order fulfilment fault, is enabled. In case of the confirmation message being received within a given timeframe, the timer is interrupted and the two elementary interactions at the bottom of the composite interaction are enabled. These two messages show the two possibilities that can happen after a transport service has confirmed the dispatch time: if the delivery can be executed without any problems, the transport service is sending an confirmation to the warehouse, otherwise the message about a delivery fault is sent. Thus, the two interactions are connected via a two-way inhibits relationship.

Pattern 12: Relayed request

- Description: Party A makes a request to party B which delegates the request to other parties (P1, , Pn). Parties P1, , Pn then continue interactions with party A while party B observes a "view" of the interactions including faults. The interacting parties are aware of this "view" (as part of the condition to interact).
- Example: Some supportive work of managing regulatory provisions outsourced by government agencies to external agencies fits this pattern. Party A is a client seeking some outcome pending regulation, e.g. obtaining particular land tenure. Party B is the government authority concerned with the regulation. e.g. lands department. Parties P1, , Pn are outsourced service providers from the government authority's regulation process, e.g. brokers who validate applications and external land management experts who can provide independent audit of applications. The government authority stipulates that interactions between the client and outsourced service providers associated with key points of processing, such as the start and end of activities, and key reports, be sent to it.

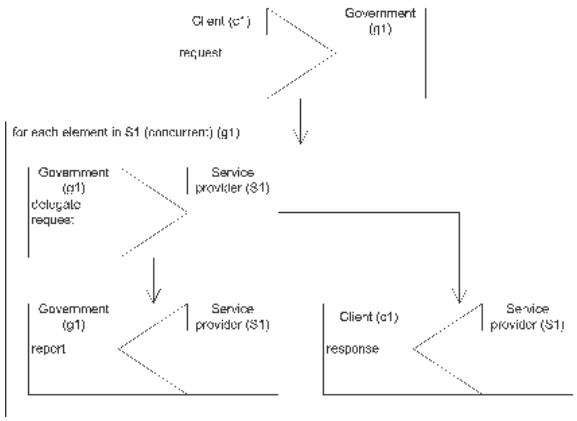

Fig. 30. Pattern 12: Relayed request - Example 1

– Diagram (Fig. 30): The above elementary interaction shows the sending of a request from a client to an actor referred to as “g1” and playing the role “Government”. This interaction enables the lower composite interaction, depicting the delegation of the request to multiple service providers. These service providers are nominated by the government agency and thus the actor referred to as “g1” has to bind the set of actor references “S1”. This is indicated by the content of the small rectangle on top of the repeated interaction. After the delegation of the request has been completed, the two remaining elementary interactions are enabled. They depict the sending of the response to the client and the sending of a report to the government agency.

Pattern 13: Dynamic routing

– Description: A request is required to be routed to several parties based on a routing condition. The routing order is flexible and more than one party can be activated to receive a request. When the parties that were issued the request have completed, the next set of parties are passed the request. Routing can be subject to dynamic conditions based on data contained in the original request or obtained in one of the “intermediate steps”.
– Example 1: Flexible order fulfillment. After processing an order, the sales department sends a request to the finance department to process the invoicing and payment receipt for the order. This request contains a reference to the customer’s procurement service and possibly also to a shipping service nominated by the customer. After arranging invoicing and payment by interacting directly with the customer, the finance service forwards the order to the warehouse service. If the order is marked “for pick-up”, the warehouse eventually sends a notification of availability for pick-up to the customer’s procurement service. Otherwise, the warehouse issues a request to a shipping service which may be either the company’s default shipping service, or the one originally nominated by the customer. The shipping service eventually sends a shipping notification directly to the customer.
– Diagram 1 (Fig. 31): The top elementary interaction shows the sending of a request from the sales department to the finance department. The actor referred to as “s1” and playing the role “Sales” is binding two actor references “c1” and “h1”. They refer to actors playing the roles of the procurement service and the shipping service of the customer respectively. The respective roles are named “Customer” and “Shipping”. This interaction enables the execution of invoicing and payment between finance department and the procurement service of the customer. If this interaction is completed, the interaction for forwarding the order to the warehouse is enabled. This interaction is connected with two precedes relationships with two elementary interactions which themselves are connected with a two-way inhibits relationship. These racing messages depict the two options after the forwarding of the order to the warehouse: if the order is marked “for pick-up”, the warehouse is sending a notification to the customer, whereby the receipt of the message is executed by the actor reference bound by the sales department.

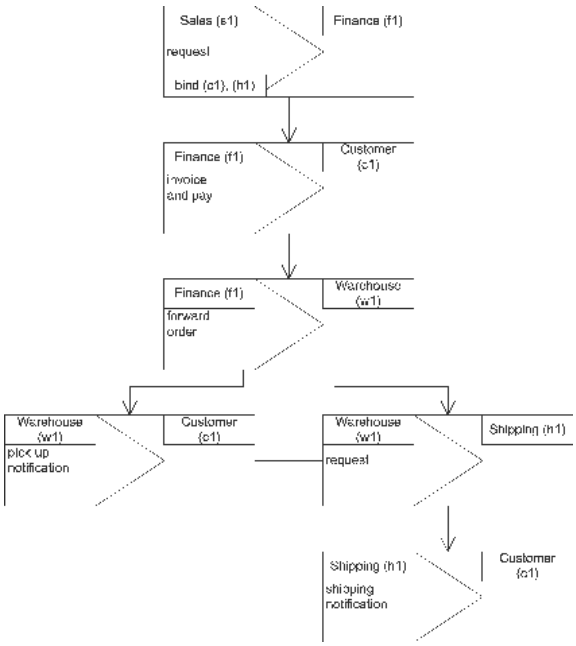

Fig. 31. Pattern 13: Dynamic routing - Example 1

Otherwise the warehouse is sending a request to the customer, whereby the receipt of the request is performed by the actor reference "h1". This interaction enables a final notification, sent from the shipping service to the customer.

- Example 2: Proposal reviews. A project proposal initiated by a project coordinator is required to be passed through its work-package coordinators in any order, one at a time. For each route, all coordinators get copies of the document, however only one, i.e. the first expressing interest, is allowed to do the update. A coordinator updates the document and makes it available for the next coordinators' "read only" copy, out of which one gets "write" access. After an update, a problem may be flagged which requires the proposal to be routed back to the project coordinator. This over-rides the next step of the routing, and a modification of the routing may be issued by the project coordinator.
- Diagram 2 (Fig. 32): The above composite interaction depicts the sending of the project proposal to all work-package coordinators, whose role is named "Coordinator". This sending is executed by the project coordinator, whereby the set of actor references referring to the set of work-package coordinators "C1" is bound by the actor referred to as "p1" and playing the role "Project coordinator". After the completion of this interaction the two remaining interactions are enabled, whereby only one of them can occur, since they are connected via a two-way inhibits relationship. The left interaction occurs if a

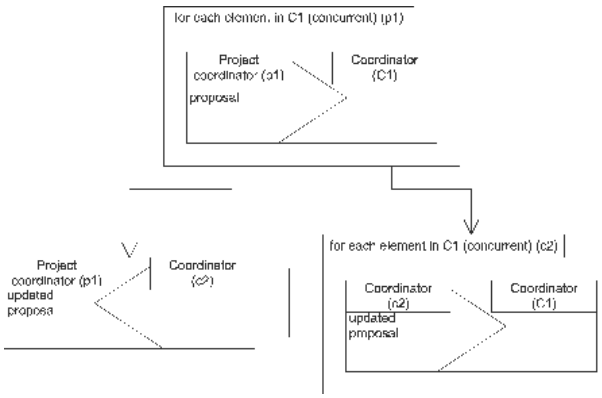

Fig. 32. Pattern 13: Dynamic routing - Example 2

problem occurs after updating the proposal. Then the work-package coordinator who has executed the update and being referred to as "c2" is sending the updated proposal to the project coordinator. If the update could be executed without any problems, the work-package coordinator who has executed the update is sending the updated proposal to all other work-package coordinators. Therefore he has to bind again the set of actor references "C1".

- Example 3: Legal case preparation. A legal case preparation service is utilized by law courts to reduce the number of hearing re-schedules. This is a costly problem for the courts and defers justice for litigants due to insufficient information to embark on hearings, decreed by judges typically within the first moments of a hearing. As part of improved preparation, the clerk of the court examines a scheduled hearing, obtains all relevant documents into a formal draft, and determines the relevant legal or administrative actors required to provide examine the draft for verification and additional input of the draft. The clerk determines the first set of actors (defense and prosecution lawyers, and courts jurisdictionally related to the case) to review the draft. After these, expert opinion is canvassed based on issues raised in the investigation, e.g. different departmental solicitors in different categories of expertise.
- Diagram 3 (Fig. 33): The depicted repeated interaction consists of two elementary interactions that are connected via a precedes relationship. The first elementary interaction shows the sending of a request from the clerk to the expert, whereby the clerk is referred to as "c1" and the expert as "e1". The clerk is binding the set of actor references for the involved lawyers ("L1") and the set of actor references for the involved courts ("T1"). If this interaction has been completed the second elementary interaction is enabled, showing the sending of a response from the expert to the clerk. Before executing this interaction, the expert is binding the sets of actor references again, which allows for the change of the two sets. The whole composite interaction is

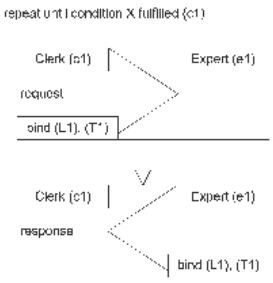

Fig. 33. Pattern 13: Dynamic routing - Example 3

repeated until a condition “X” is fulfilled, which is evaluated by the actor referred to as “c1”.