Design and Analysis of Process Choreographies

Gero Decker

Business Process Technology Group
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
— Dr. rer. nat. —

— Doctoral Thesis —

June, 2009

Zusammenfassung

Elektronische Integration zwischen Organisationen erfordert eine präzise Spezifikation des Interaktionsverhaltens: Informationssysteme, die Kommunikation per Telefon, Fax und Email ablösen, können nicht so flexibel und selbständig auf Ausnahmesituationen reagieren wie Menschen. Choreographien ermöglichen es, Interaktionsverhalten genau zu spezifizieren. Diese Modelle zählen die beteiligten Rollen, die erlaubten Interaktionen, Nachrichteninhalte und Verhaltensabhängigkeiten auf und dienen somit als Interaktionsvertrag zwischen den Organisationen. Auch als Ausgangspunkt für eine Anpassung existierender Prozesse und Systeme sowie für die Implementierung neuer Softwarekomponenten finden Choreographien Anwendung.

Da ein Vergleich von Choreographiemodellierungssprachen in der Literatur bislang fehlt, präsentiert diese Arbeit einen Anforderungskatalog, der als Basis für eine Evaluierung existierender Sprachen angewandt wird. Im Kern führt diese Arbeit Spracherweiterungen ein, um die Schwächen existierender Sprachen zu überwinden. Die vorgestellten Erweiterungen adressieren dabei Modellierung auf konzeptioneller und auf technischer Ebene.

Beim Verlinkungsmodellierungsstil werden Verhaltensabhängigkeiten innerhalb der beteiligten Rollen spezifiziert und das Interaktionsverhalten entsteht durch eine Verlinkung der Kommunikationsaktivitäten. Diese Arbeit stellt einige "Anti-Pattern" für die Verlinkungsmodellierung vor, welche wiederum Untersuchungen bzgl. Choreographiesprachen des Interaktionsmodellierungsstils motivieren. Hier werden Interaktionen als atomare Blöcke verstanden und Verhaltensabhängigkeiten werden global definiert. Diese Arbeit führt zwei neue Choreographiesprachen dieses zweiten Modellierungsstils ein, welche bereits in industrielle Standardisierungsinitiativen eingeflossen sind.

Während auf der einen Seite zahlreiche Fallstricke der Verlinkungsmodellierung umgangen werden, können in Interaktionsmodellen allerdings neue Anomalien entstehen. Eine Choreographie kann z.B. "unrealisierbar" sein, d.h. es ist nicht möglich interagierende Rollen zu finden, die zusammen genommen das spezifizierte Verhalten abbilden. Dieses Phänomen wird in dieser Arbeit über verschiedene Dimensionen von Realisierbarkeit untersucht.

Abstract

With the rise of electronic integration between organizations, the need for a precise specification of interaction behavior increases. Information systems, replacing interaction previously carried out by humans via phone, faxes and emails, require a precise specification for handling all possible situations. Such interaction behavior is described in process choreographies. Choreographies enumerate the roles involved, the allowed interactions, the message contents and the behavioral dependencies between interactions. Choreographies serve as interaction contract and are the starting point for adapting existing business processes and systems or for implementing new software components.

As a thorough analysis and comparison of choreography modeling languages is missing in the literature, this thesis introduces a requirements framework for choreography languages and uses it for comparing current choreography languages. Language proposals for overcoming the limitations are given for choreography modeling on the conceptual and on the technical level.

Using an interconnection modeling style, behavioral dependencies are defined on a per-role basis and different roles are interconnected using message flow. This thesis reveals a number of modeling "anti-patterns" for interconnection modeling, motivating further investigations on choreography languages following the interaction modeling style. Here, interactions are seen as atomic building blocks and the behavioral dependencies between them are defined globally. Two novel language proposals are put forward for this modeling style which have already influenced industrial standardization initiatives.

While avoiding many of the pitfalls of interconnection modeling, new anomalies can arise in interaction models. A choreography might not be realizable, i.e. there does not exist a set of interacting roles that collectively realize the specified behavior. This thesis investigates different dimensions of realizability.

Acknowledgements

In the course of working on my PhD thesis I was lucky to be surrounded by a number of very inspiring fellow researchers who constantly provided feedback and input. By chance, I got involved in a choreography research project in 2006. The project was jointly carried out by SAP Research and the Queensland University of Technology. Thanks, Alistair Barros, Marlon Dumas, Johannes Maria Zaha and Arthur ter Hofstede for teaching me how to do solid research and for pushing me towards the interesting questions in the field of choreographies.

Especially the collaborations with Oliver Kopp and Frank Leymann from the University of Stuttgart, Niels Lohmann from the University of Rostock and Jan Mendling from the Humboldt University of Berlin resulted in significant steps forward. Oliver guided me in the technicalities of web services, Niels was an excellent sparring partner for all formal issues and Jan even introduced me to the world of empirical research.

My fellow researchers of the Business Process Technology group also deserve a big thanks for inspiring discussions and plenty of feedback. Especially making the first research steps with Frank Puhlmann, writing the BPMN-book *The Process* [117] with Alexander Grosskopf and working on BPM papers with Hagen Overdick, Matthias Weidlich and Ahmed Awad have been a big motivation. Thanks to the great work of all contributors of the Oryx project. This enabled me to very efficiently realize my software prototypes and save precious time. I would also like to thank my reviewers Gregor Engels and Wil van der Aalst for their extensive feedback on my thesis and the reviewers at conferences and workshops for their many insightful reviews.

Last but not least I want to thank Mathias Weske, my PhD supervisor, who gave me all the support imaginable in pursuing my research work. His continuous feedback and encouragement were a vital part in the endeavor of my PhD thesis.

Contents

Chapter 1

Introduction

With the rise of enterprise information systems, significant improvements of business operations are possible. Central storage of data and access to it from different locations change the way how business activities are designed. Enterprise information systems such as Enterprise Resource Planning systems and Customer Relationship Management systems were introduced [219]. Based on the capabilities of these systems, handover of work between persons and the distribution of responsibilities within organizations are revisited. Business processes being collections of business activities performed to realize a business goal, moved into the center of attention. In a wave of business process reengineering whole organizations were restructured, business processes were designed and improved performance of business processes was aimed at, often inspired by the possibility offered by enterprise information systems [121, 63].

While the first wave of information systems focused on the activities within organizations, a second wave of systems and initiatives target cross-organizational integration. Traditional means of communication between organizations such as phone calls, letters and faxes are partly replaced by electronic message exchanges using standard protocols and message formats, such as Electronic Data Interchange [8]. This avoids delays and reduces human involvement as well as the number of errors made in the transformation between different media.

Electronic communication is both applied in the area of business-to-business (B2B) integration and in the business-to-consumer (B2C) area. The latter results in higher availability of the company and better consumer experience. The former results in huge productivity gains and innovation in domains such as supply chain management, financial services and logistics [219].

Especially in the case of B2B integration, outsourcing of individual activities or even of whole business processes becomes easier. The creation of distributed value chains – where different organizations contribute only those activities for which they provide the highest quality, productivity and innovation – can be done faster and with less cost in a world of electronic integration.

Moving away from human interactions as realization of cross-organizational integration imposes new challenges. While previously only a rough understanding of the business processes involved were needed (any ambiguities could be resolved by the employees involved), electronic interaction requires a much more careful design of the interaction protocols used. The information systems need to be configured to be able to also react correctly in exceptional cases.

To facilitate a high degree of automation, it has to be decided where information exchange between the organizations is required. Then, in turn, information exchange needs to be further refined to come to a message exchange level, where documents with defined business meaning and defined message format come into play [94]. This is due to the fact that machines need much more precise definitions of what they are supposed to do than humans.

Furthermore, the desired order of message exchanges and especially reactions to cancellation at different points during the interaction must be agreed upon by all interacting organizations before actually carrying out the interaction.

Process choreographies are a means to represent and manage cross-organizational interaction, even in multi-lateral settings. Choreographies are models that enumerate the organizations involved, the overall business goal to be achieved and finally the messages exchanged together with the obligations and constraints the organizations have to obey to. As it is stated in the W3C Glossary: "*A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.*" [2]. Choreographies serve as contract between the organizations and can be used as starting point for the individual information systems' configurations. Once interaction is carried out, they can be used for verifying whether all organizations behave correctly [95].

With the uptake of Business Process Management (BPM) and Service-oriented Architectures (SOA), choreographies and choreography modeling, i.e. the creation of choreographies, have recently moved into the center of attention of academic research and industry initiatives. This thesis aims at providing a consolidated view on the state-of-the-art in choreographies and addresses a number of novel issues that are crucial for the adoption of choreography modeling also in industrial practice. Therefore, this thesis not only answers conceptual questions around choreographies but also gives an overview of the challenges behind choreography modeling. This thesis concentrates on behavioral aspects of choreographies on the one hand. On the other hand, it investigates the relationship between choreographies and business processes that are internal to individual organizations.

In order to facilitate the discussion of the different concepts covered, the next section will introduce an example. It will be used for illustration throughout this thesis. The most important choreography terminology will be introduced in Section 1.2. It provides the vocabulary for this thesis. Section 1.3 describes the problem statement. Finally, the contributions of this thesis will be pointed out and the outline of this thesis will be presented in Sections 1.4 and 1.5.

1.1 Motivating Example

Auctioning brings together a *seller* and a *buyer*. The seller owns a certain good or offers a certain service. There are a number of potential buyers interested in the same item offered. The auctioning mechanism determines the actual buyer by allowing *bidders* (the potential buyers) to place bids. The auction has a predefined timeframe, e.g. several days. Only during the auction, the bidders are allowed to place their bids. The seller might define a minimum bid and there might be a fixed bidding increment depending on the currently highest bid and the policies of the auctioning service. Once the auction is over, the bidder who has placed the highest bid wins and follow-up activities such as payment and delivery are initiated.

Online auctioning is a very popular and efficient realization for the auctioning mechanism. Here, a software system coordinates the auctioning process. The platform allows to upload the description of an item and offers search functionality for people interested in a certain item. The seller and the bidder are typically people who access the auctioning platform through their web browser and use the offered functionality for creating auctions, placing bids and finally completing the payment. Alternatively, software systems might also realize (parts of) the behavior of sellers and bidders. This increases the efficiency for high-volume sellers or bidders.

For most auctions, there is no human involvement necessary from the platform provider's side. For these cases, all functionality is implemented as software systems running 24 hours a day. In addition to the regular cases, exceptions might occur before, during or after auctions. Some exceptions require human involvement, others are also handled by software systems. Sellers might offer items that are not allowed to be traded, e.g. human organs, child pornography or Nazi propaganda material. In addition, the platform provider might not accept certain items, such as designer bags, as the danger of replicas would be too high. Companies might request auctions to be canceled if they detect fraudulent offers regarding their own products.

Other exceptions involve problems between sellers and buyers after the acceptance of the contract of sale. E.g. the buyer has transferred the specified amount of money to the seller but the seller does not send the purchased item. In this and other cases, buyers and sellers make use of the platform provider's customer support services. While some requests can be handled in a fully automatic way, many requests need to be handled by one of the customer support agents.

In addition to sellers, bidders and the auctioning service there might be further participants involved. For instance, the selling part can be outsourced. Such offerings might be interesting for those sellers who are unexperienced with online auctions or simply do not have the time to care about it. Another group of parties involved are shippers who take care of the delivery of the goods. Also the usage of external payment services is very common for online auctions.

This thesis will mainly concentrate on the core auctioning process, where three roles are involved: seller, bidder and auctioning service. Four main phases can be distinguished. First, the auction setup phase takes place. Then, the actual auction phase happens where bidders can place their bids. Finally, payment and shipment are carried out.

Each phase can be further refined into message exchanges. For instance, a bid message, a bid acknowledgment message and finally notification messages telling each bidder whether she has won the auction or not and giving details about the buyer to the seller.

1.2 Choreography Terminology

In order to stick to a consistent terminology throughout this thesis, this section introduces the general concepts of choreographies and puts them into relation. The auctioning domain will be used to illustrate the terms.

Real-world organizations, persons, information systems or software services that interact with other organizations, persons, systems or services are called *participants*. Examples would be Gero Decker, ebay Inc, the server addressed by 141.89.225.120 or the web service hosted at http://example.com/MyService. *Roles* are models of participants, each describing a set of participants that show a certain similarity in the context of the domain under consideration. For

instance, “auctioning service” could be the corresponding role for the participants ebay Inc and Christie’s Inc or the role “seller” for the participant Gero Decker.

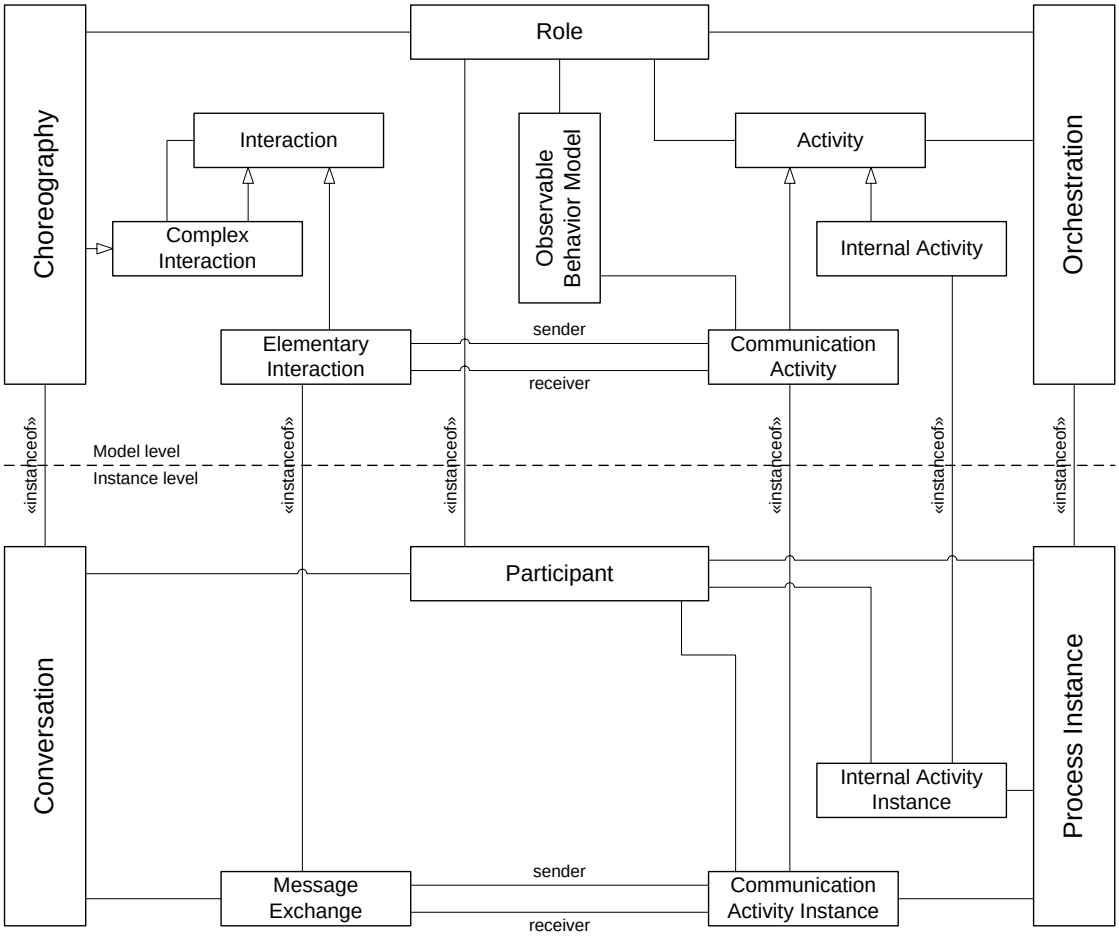


Figure 1.1: Choreography Terms

Two participants interact using *message exchanges*. For instance, Gero Decker sends an email to ebay Inc. The counterpart on the model level is an *elementary interaction* between two roles, e.g. a message creation request sent by a seller to an auctioning service. *Complex interactions* are composed of a set of interactions, e.g. the collection of requests and responses sent between the seller and the auctioning service until an auction is created. In the other direction, complex interactions can be refined into elementary interactions.

Concrete actions performed by participants are called *activity instances*. They can be further divided into *internal activity instances* and *communication activity instances*. The latter are related to message exchanges while the former are not observable by other participants. Gero Decker opening his word processing program would be an internal activity instance, while Gero Decker hitting the submit button of his email program would be a communication activity instance (provided that the computer and the programs installed on it are not considered as

participants in the domain of consideration).

Activities, also divided into *internal activities* and *communication activities*, are located on the model level. For instance, the act of sending an auction creation request performed by a seller is a communication activity, while storing the request would be an internal activity performed by an auctioning service.

Choreographies are complex interactions with behavioral dependencies between the contained interactions. At least two roles are involved in one choreography. *Conversations* are choreography instances, i.e. sequences of message exchanges. The allowed sequences of message exchanges in a conversation are constrained through the corresponding choreography.

Collections of activity instances performed by the same participant are grouped into *process instances*. On the model level, *orchestrations* consist of activities performed by the same role, together with behavioral dependencies between the activities. Again, the allowed execution orders of activity instances in a process instance are constrained through the corresponding orchestration.

Hierarchy of roles is largely ignored in the context of choreographies. For instance, the fact that a seller could be partitioned into a warehouse, a logistics department and a sales department is not considered. The focus of attention lies on the seller's interaction with the environment. A role is qualified through the decision that its interaction behavior regarding an environment is of interest for the specific choreography modeling purpose. In contrast to this, hierarchy of roles is an important aspect in orchestrations. Here, organizational units, job types, individual employees or subsystems are considered.

The main difference between choreographies and orchestrations is that choreographies focus on interactions between roles, while orchestrations focus on the central coordination of activities. A choreography is not executed by a central coordinator. It is rather a specification of the interaction behavior among a set of independent roles. An orchestration, in contrast, can be executed and controlled by a central orchestrator. The question of whether a model is a choreography or an orchestration can be answered best through an execution perspective. If the coordination among all activities contained in the model is carried out by a central machine or person, the model is an orchestration. If the model contains a number of roles that act independently, the model is a choreography. Thus, a modeling language might allow to represent orchestrations as well as choreographies. However, a good indication that a model is a choreography is whenever there exists a clear distinction between behavioral dependencies within a role on the one hand and interactions between roles on the other hand.

An *observable behavior model* can be seen as intersection between a choreography and an orchestration: It contains communication activities related to the same role and therefore describes the observable behavior of that role.

Figure 1.1 illustrates these terms and their relationships using the UML class diagram notation [5]. Choreography (and orchestration) languages would reside on a third level, the metamodel level, defining the rules for combining interactions and roles into choreographies (or for combining activities into orchestrations).

In the literature, *global model* or *conversation model* is sometimes used as synonym for choreography and *local model* or *interface process* as synonym for observable behavior model. [95] elaborates on these different viewpoints and establishes a relationship between them. A

similar distinction between orchestrations and choreographies like presented in this section can also be found in [214].

1.3 Problem Statement

The motivating example showed that complex interaction behavior between multiple roles must be considered. While the description in Section 1.1 remained on a high level of abstraction, it is easy to see that a huge number of interactions will appear during refinement and that they will be related by complex dependencies.

Textual descriptions of complex interaction behavior are not sufficient. Text cannot properly reflect the design decisions that have to be made for electronic integration of different partners. A number of questions regarding the example remain. Should the payment only be carried out after the goods have arrived? Or should it be the other way round? Are bidders allowed to place several bids or just one? What kind of documents need to be exchanged in the course of an auction? Can an auction be canceled? What happens if no bidder is interested in the good – does the seller still need to pay for the auctioning service? Is every bidder going to be notified about the outcome of the auction? When does the seller first get into contact with the successful bidder? Informal descriptions leave ambiguities. Only models with a precise meaning avoid these ambiguities. Modeling languages provide a common vocabulary to facilitate communication.

Despite its significant overlap with modeling orchestrations, choreography modeling comes with a number of specifics that deserve special attention. The additional dimension of interaction between different partners poses new challenges.

Question 1: What is a suitable choreography language? While orchestration languages have been studied extensively in the literature, the field of choreography modeling only recently enjoys attention by the academic community. Industry projects have a pressing demand for suitable choreography languages. Three basic requirements for choreography languages can be identified.

1. *Expressiveness.* The previous section has already presented a number of concepts from the choreography domain. It must be possible to express typical conversations using a choreography language. Furthermore, the constructs of the language need unambiguous semantics, in order to avoid misunderstanding and confusion.

2. *Ease of use.* In addition to the mere expressiveness, it should only require few modeling elements to express the most common scenarios. This is important for modeling speed and for the readability of models. On top of that, a choreography language should be designed in such a way that modeling errors are unlikely and that typical decision making processes are supported through the language.

3. *Ease of adoption.* Human modelers are confronted with a wide range of modeling languages. The adoption of a choreography language highly depends on how close it is to existing languages in terms of concepts / semantics supported and notational elements

used. A choreography language ideally does not deviate much from existing and widely used modeling languages.

The design of choreography languages is one important pillar for enabling choreography modeling. A second pillar centers around model quality. Within this area, ensuring the absence of modeling errors is of key importantance. While modeling errors in orchestrations have been studied extensively, the focus on interactions, as it is present in choreographies, imposes novel challenges.

Question 2: Which choreographies can be implemented by a set of participants? A fundamental difference between choreographies and orchestrations is that we do not assume a central coordinator in the case of choreographies. All interacting participants are autonomous. They know their internal state and can only make assumptions about the other participants' states through the messages sent and received. This might result in situations where the behavior specified in choreographies cannot be *realized* by interacting participants.

Furthermore, the interplay between several process instances in a conversation might lead to anomalies that are not present in orchestrations. Instead of centralized execution control, the realization of choreographies relies on inter-related message exchanges. In this context, missing isolation between conversations might in turn lead to undesired behavior, i.e. behavior that is not allowed by the choreography. On the quest for techniques that verify that a given choreography can be implemented properly, we see the following two requirements.

1. *Automatic verification and location of errors.* If a choreography with defined semantics is provided, it must be possible to perform fully automatic verification. This is important as choreographies easily grow very complex and manual verification becomes cumbersome and error-prone. Furthermore, it is not sufficient to detect that a choreography is not correct. Errors must be located so that the modeler gets a hint about where changes need to be applied to the model.

2. *Reuse of existing formalisms.* Automatic verification requires a formal model and in order to apply formal models, translations of the choreography languages must be present. Therefore, reuse of existing formal models is desirable as it allows to reuse/extend existing translation approaches as well as existing tools and proofs.

1.4 Scientific Contribution

According to the two questions identified in the previous section, this thesis centers around suitability and design of choreography languages on the one hand and formal verification regarding correctness of choreographies on the other.

A thorough analysis and comparison of choreography languages is missing in the literature. Therefore, this thesis puts forward a set of requirements for choreography languages. A distinction is made between choreography modeling on the conceptual level and choreography modeling on the technical level. The requirements framework helps to answer the question

of expressiveness for existing choreography languages. It also incorporates requirements derived from common choreography design methods, revealing abstraction mechanisms needed for properly reflecting and relating the modeling decisions made in a choreography project. By applying this requirements framework to existing choreography languages, open issues are identified and directions for improvement are highlighted.

Based on the results from this assessment, four proposals regarding language extensions and novel choreography languages are made. Two proposals follow a conservative approach, adding as little extensions to existing languages as possible.

1. *BPMN extensions.* By refining the data object concept of the Business Process Modeling Notation (BPMN [9]) and introducing multiplicity of roles and complex message flow, open issues in current BPMN can be overcome and a broader range of conversations can be reflected properly.

2. *BPEL4Chor.* The Business Process Execution Language (BPEL [108]) is an orchestration language. In this thesis, BPEL is extended with a topology concept and direct support for participant sets, shifting it to a choreography language.

The two other proposals follow a more radical approach.

1. *Let's Dance.* Based on recurring conversations in real-world scenarios, which also form an important part of the requirements framework, are used as input for designing an entirely new choreography language. New notational elements and novel semantics are introduced.

2. *iBPMN* is a compromise between novel semantics as present in Let's Dance and known notational elements from BPMN. Atomic interactions form the basic building blocks and behavioral constraints are defined between them from a global perspective, as it is the case in Let's Dance. In contrast to this, the constructs for expressing the behavioral dependencies are taken from BPMN.

Integration between the different approaches is investigated and first practical insights into the adoption of the proposals are reported. Some of the ideas developed in the context of this thesis have already made their way into standards proposals, especially version 2.0 of the Business Process Modeling Notation (BPMN [164]) which will be released through the Object Management Group (OMG)[1]. It contains a dedicated choreography profile which was inspired by Let's Dance and iBPMN as presented in this thesis.

A survey of existing choreography formalisms and correctness criteria reveals that a wide range of issues have already been tackled in the literature, specifically ensuring the absence of deadlocks in choreographies (compatibility, controllability) and ensuring that a refinement of a specification will interact successfully with other participants (conformance). These are useful techniques when checking whether a choreography can be implemented properly by a set of interacting participants.

[1] See `http://www.omg.org/`

However, it turns out that most techniques only consider individual conversations. The additional anomalies occurring in the case of multiple concurrent conversations has been neglected so far. Therefore, this thesis proposes a novel correctness criterion for choreography, namely *instance isolation*. Based on a formal model inheriting concepts of place/transition nets and π-calculus, this technique ensures that multiple process instances of the same role will not interfere.

In the course of this thesis, a distinction between between the *interconnection modeling style* and the *interaction modeling style* is made. While interconnection models distinguish send and receive activities and define behavioral constraints on a per-role basis, interaction models consist of atomic interactions that are related through global behavioral constraints. The behavioral constraints are global in the sense that they are not explicitly assigned to any role by the modeler. This allows to represent constraints that are not enforceable by any of the roles involved.

The notion of *realizability* helps do detect whether there exists a set of interacting participants that respect all behavioral constraints defined in an interaction model. This thesis reveals multiple dimensions of the realizability notion, leading to the correctness criteria *full realizability*, *local enforceability* and *desynchronizability*. While having a thorough mathematical background, the results are practically applicable as they ultimately allow to detect and locate modeling errors in choreographies.

1.5 Outline of this Thesis

This thesis is divided into seven chapters. This first chapter has motivated the need for choreography modeling in general and suitable choreography languages in particular. The next chapter reports on related work. It introduces the areas of Business Process Management (BPM) and Service-oriented Architectures (SOA). Furthermore, existing choreography languages, choreography formalisms and correctness criteria for choreographies are presented.

Chapter 3 evaluates the related work. Especially, the choreography languages are assessed based on a requirements framework for choreography languages that is derived from existing choreography design approaches, the motivating example from Section 1.1 and existing comparison frameworks. It distinguishes choreography modeling on the conceptual level and choreography modeling on the technical level. The assessment of the choreography languages along this framework leads to the identification of open issues. In this context, the distinction between an *interconnection modeling style* and an *interaction modeling style* is made, providing the structure of the two main chapters of this thesis.

Chapter 4 investigates the interconnection modeling style and presents BPMN extensions as improved choreography language for the conceptual level and BPEL4Chor as novel choreography language for the technical level. The issue of proper correlation configurations in choreographies is investigated, leading to the property of *instance isolation*. This property ensures that two process instances of the same orchestration are not involved in the same conversation. Instance isolation checking is based on a special class of Petri nets, so called ν^*-nets, which incorporate name passing and name creation capabilities.

Chapter 5 addresses the interaction modeling style. A formal model based on Petri nets is established and Let's Dance and iBPMN are presented as novel choreography languages for

the conceptual level. Formal semantics is provided for both languages using a transformational approach. Several correctness notions of realizability are introduced, including *full realizability*, *local enforceability* and *desynchronizability*.

Chapter 6 reports on practical validation of the approaches presented in this thesis through software implementation. A tool suite based on the Open Source platform Oryx[2] implements choreography modeling capabilities for the different language proposals, as well as formal verification regarding the novel choreography properties.

Finally, Chapter 7 concludes this thesis. The limitations of the contributions are discussed and open issues are pointed to.

1.6 Publications

Many of the contributions of this thesis have already been published in national and international conferences and workshops as well as in international journals.

General introductions to choreographies were published in the Informatik Spektrum journal [65] and in the it – Information Technology journal [73].

Section 2 contains results that were published in the following papers. The applicability of π-calculus and colored Petri nets for formalizing the Service Interaction Patterns was investigated in [85], which was presented at the International Conference on Business Process Management (BPM). A formal mapping of the Business Process Execution Language (BPEL) to π-calculus was presented in [213] at the Asia-Pacific Services Computing Conference (APSCC). Executability of high-level Petri nets was studied in [78] and presented at the Workshop on Web Services and Formal Methods (WS-FM). Different correctness notions for choreographies were surveyed and compared in [88], a paper that was presented at the International Conference on Advanced Information Systems Engineering (CAiSE). A verification approach for business process models was presented in [30] at the International Conference on Business Process Management (BPM). Bottom-up choreography design approaches were investigated in [87], joint work with the University of Hamburg that was presented at the International Conference on Business Information Systems (BIS). A top-down choreography design approach was presented in [34] at the Workshop on Software Engineering Methods for Service Oriented Architecture (SEMSOA), resulting from a close collaboration with SAP Research and the Queensland University of Technology.

An evaluation of the Web Services Choreography Description Language (WS-CDL) regarding its suitability for choreography modeling was investigated in [83] which was presented at the EMISA workshop. An in-depth survey and assessment of existing choreography languages is also part of [75]. These results were integrated into Section 3. A more detailed elaboration on one of the Service Interaction Patterns can be found in [37], which was presented at the Workshop on Trends in Enterprise Architecture Research (TEAR). The relationship between the Business Process Modeling Notation (BPMN) and BPEL was investigated in [212] and presented at the International Conference on Cooperative Information Systems (CoopIS). BPMN's relationship to YAWL was investigated in [70] and presented at the Demo Session of the Inter-

[2] See `http://code.google.com/p/oryx-editor/`

national Conference on Business Process Management (BPM). Further investigations on BPMN 1.1 can be found in [86].

Section 4 is based on the following papers. The BPMN extensions from Section 4.1 have been published in [84] at the International Conference on Cooperative Information Systems (CoopIS). BPEL4Chor resulted from a close collaboration with the University of Stuttgart and was first presented in [75] at the International Conference on Web Services (ICWS). An extended paper [76] was published in the Data & Knowledge Engineering journal (DKE). Further investigations on BPEL4Chor have been published in [77] at the European Young Researchers Workshop on Service Oriented Computing (YR-SOC), in [176] at the International Workshop on Engineering Service-oriented Applications: Analysis, Design and Composition (WESOA) and in [74] at the International Conference on Advanced Information Systems Engineering (CAiSE). Typical correlation scenarios in business processes have been investigated in [35] and were presented at the International Conference on Fundamental Approaches to Software Engineering (FASE). Instance isolation analysis was presented in [90] at the International Conference on Services Computing (SCC). The work on correlation also relates to investigations on instantiation semantics for business process models as presented in [79] at the International Conference on Business Process Management (BPM) and an extended version in [80] which appeared in the Data & Knowledge Engineering journal (DKE). The work on process instantiation resulted from a collaboration with the Humboldt University of Berlin.

Several papers have presented and investigated the choreography language Let's Dance. The language resulted from joint efforts with the Queensland University of Technology and SAP Research and was first introduced in [220] at the International Conference on Enterprise Distributed Object Computing (EDOC). Formal semantics was presented in [91] at the Workshop on Web Services and Formal Methods (WS-FM) and a graphical editor and analysis tool for Let's Dance in [72] at the Demo Session of the 4th International Conference on Business Process Management (BPM). A journal paper on Let's Dance, including investigations on local enforceability and a mapping to BPEL appeared in the IEEE Transactions on Systems, Man, and Cybernetics, Part C [221]. iBPMN, jointly developed with SAP Research, was first presented in [68] at the International Workshop on Collaborative Business Processes (CBP). The issue of realizability and local enforceability of choreographies was studied in [89] and was presented at the International Conference on Business Process Management (BPM). The notion of desynchronizability, a result of joint work with the University of Rostock and SAP, was presented in [69] at the International Conference on Service Oriented Computing (ICSOC). An overarching paper on realizability of interaction models with its different dimensions was presented in [66] at the Central European Workshop on Services and their Composition (ZEUS).

Oryx served as basis for the prototypical implementation for the approaches presented in this thesis and was described in several papers. Among them are [81] at the Demo Session of the International Conference on Business Process Management (BPM), [82] at the Demo Session of the International Conference on Conceptual Modeling (ER), [71] at the EMISA workshop, [145] at the EPK workshop and [67] in the Computer Zeitung journal.

Chapter 2

Related Work

As shown in the motivating example and the choreography terminology section, choreographies center around the notions of *roles*, *interactions* and *behavioral dependencies*.

Behavioral dependencies are heavily studied in the area of Business Process Management (BPM). Business process models describe how a business goal is achieved through a number of activities. In most cases, business process models are orchestrations, where all activities are carried out within the same organization. BPM is at the same time a management discipline and an enabling technology to realize a process-oriented view on organizations. Section 2.1 will introduce BPM in more detail.

Interactions between independent systems are at the center of attention in Service-oriented Architectures (SOA). From a technical perspective, SOA is an architectural style for interconnected software systems with a strong focus on message exchanges. The internals of the software services are irrelevant as long as they behave as specified in the observable behavioral models. Section 2.2 will introduce the main concepts of SOA. BPM and SOA are heavily used in practical settings. Therefore, process choreographies are not only a topic studied in the academic arena. They have high relevance for practical challenges.

A number of modeling languages have been proposed in the academic literature and by standardization initiatives in the BPM and SOA space. Section 2.3 presents the most prominent languages supporting the concepts of roles, interactions and behavioral dependencies.

In order to allow rigor and avoid ambiguities, formal models provide a solid basis for discussing the semantics of choreographies. Section 2.4 gives an overview of existing formal models. Based on that, typical correctness criteria for observable behavior models and choreographies are highlighted in Section 2.5.

2.1 Business Process Management

Business processes are in place since humans do business. Products or services are offered to customers in exchange for other products, services or money. Several activities need to be performed in order to create the products or to provide the service. In this context, distribution of labor is a central aspect. The insight that different people have different talents and skill sets provides the basis for assigning different activities to different people.

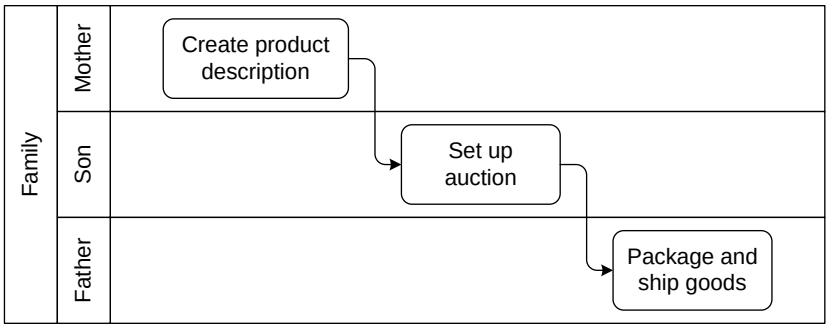


Figure 2.1: Simple orchestration

As simple example you could imagine a family selling old child clothes through an online auctioning platform. The goal is to sell the clothes for the highest possible price. As one of the activities, somebody needs to create a product description including a photo and a textual description. The mother is very good at taking a nice picture and writing an attractive text. The son knows the auctioning platform very well and is the fastest to set up the auction. The father finally takes care of packaging the clothes and taking them to the local post office.

In this example you already find some of the dimensions of business processes. (1) The *functional dimension* describes all activities that are to be performed in a business process. (2) The *control flow dimension* covers the behavioral dependencies between the activities. E.g. the product description has to be created before the auction can be set up. (3) The *organizational dimension* introduces the people, roles or organizational units involved. (4) Different tools or machines might be available for supporting the activities. E.g. the availability of a camera is essential for taking a picture. These tools, including software systems, are mentioned in the *technical dimension*. (5) In some business processes a fifth dimension comes into play: The *data flow dimension*. Here, it is described which information is being produced or consumed by an activity. E.g. the product description is used during the auction setup activity.

The explicit documentation of business processes helps to uncover weaknesses in the current organization of activities and serve as starting point for process optimization initiatives. On the other hand, process documentation serves educational purposes—new employees entering the organization can quickly take up how things are done or, during organizational change programs, it can be shown how activities should be carried out in the new way.

Figure 2.1 depicts the example using the Business Process Modeling Notation (BPMN [6]). BPMN is the de-facto standard for graphical business process modeling. It comes with a rich set of constructs for expressing concepts from all five dimensions. A more in-depth introduction of and discussion on BPMN can be found in Chapter 3. Other prominent process modeling languages are Event-driven Process Chains (EPC [137, 192]) and UML Activity Diagrams [102].

Business Process Management (BPM) as a management discipline provides a process-oriented view on organizations. Important business processes are explicitly documented and are subject to continuous improvement. The process perspective allows to monitor and optimize business processes from end-to-end, meaning that value creation and process performance are considered across departments and functions rather than considering departments or functions

individually. That way, a global optimum is targeted rather than a number of local optima (that might even contradict each other).

With the rise of BPM as a management discipline in the 80ies and early 90ies, software systems supporting BPM initiatives have emerged. First, process modeling tools appeared that were later extended to process analysis tools. In a second step, more sophisticated information systems emerged that support the actual execution of business processes.

Especially information-centric activities can be supported by information systems. In the simplest form, they take care of managing the data produced and consumed by these activities. Other activities might not require human involvement and are fully automated.

Information systems might also help to organize the individuals' tasks and remind them of work to be finished. They suggest which activities should be performed as follow-up for some completed work. In the strongest form, information systems take care of process coordination—assigning the different work items to employees and constantly tracking the status of the process execution. The latter are called workflow management systems. These systems not only include a business process model repository and execution engine but also consist of worklist clients, adapters to other information systems and monitoring tools. The workflow reference model released by the Workflow Management Coalition (WfMC [1]) defines the components of a workflow management system [128].

Workflow management systems are the main enabler for the "Third Wave" of Business Process Management [199]. In the first wave of BPM, which already began in the 1920ies, work procedures for manual activities were formally documented. The second wave, beginning in the late 1980ies, centered around information-system-driven business process optimization. Business process models served as requirements specifications for software development, leading to software systems directly executing (parts of) the business process. In the third wave, business engineering, i.e. business process change, is to be decoupled from software engineering. This is only possible through workflow management systems that are generic software systems interpreting orchestrations. The main idea is to give more control of the systems to the process experts, the domain experts engineering the business. This gives organizations the chance to adapt to changing business environments more quickly and to innovate in shorter cycles.

Business process models, and orchestrations in particular, are the most important artifacts in Business Process Management. Different phases can be identified when working with these models—especially when they are to be executed by workflow management systems. During the *design phase* the orchestration is created, covering the five dimensions. It is important to verify and validate the orchestration. Verification checks for certain anomalies in the model. E.g. the control flow might be specified in such a way that situations can be reached, where a required synchronization of activities will never happen. Such situations are called deadlocks. Validation, on the other hand, ensures that the model actually represents what it was meant to represent.

During the *deployment phase* the individual information systems involved in the business process execution must be set up. This includes the workflow management system as well as information systems supporting individual activities or automatically carrying out activities. The technical dimension of the business process needs to be specified. Finally, the now fully con-

[1] See `http://wfmc.org/`

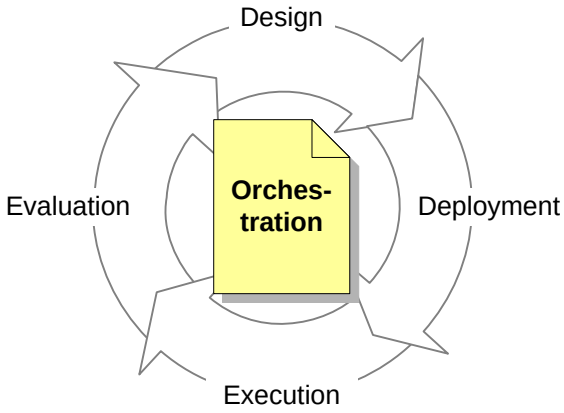


Figure 2.2: Orchestration Lifecycle

figured orchestration is fed into the workflow management system. In parallel to these software configuration tasks, the new business process must be implemented in the organization, i.e. employees need to be trained and new organizational dependencies need to be established.

Business process operation takes place in the *execution phase*. Here, process instantiation happens. Many process instances might be executed in parallel for the same orchestration. Monitoring functionality is available during execution. Statistics such as "how many auctions have completed today?" and "what is the average delay between completion of the auction and payment?" might be provided by the system. It might also be possible to zoom into individual process instances, e.g. a call center agent might handle a call from a customer who complains about a certain incident.

During the *evaluation phase* the execution history of the process instances is examined and requirements for process changes are gathered. Certain bottlenecks might have emerged or initially defined performance objectives were not met. In other settings, where workflow management systems are not in place, execution logs of information systems might be used for discovering the orchestration a posteriori. This field of research is called process mining and details can be found in [19]. The results of this fourth phase are fed back into a new design phase.

The orchestration lifecycle and its four phases are illustrated in Figure 2.2. It is a slightly adapted version from [15].

2.2 Service-oriented Architecture

The term "service" has been around in the business world for many years and has been adopted in the software world. In the area of economics a service is "the non-material equivalent of a good". A service in the software world has similar characteristics. A software service can be requested through electronic messages. Services are loosely coupled components described in a uniform way that can be discovered and composed [2]. They support rapid, low-cost composition of dis-

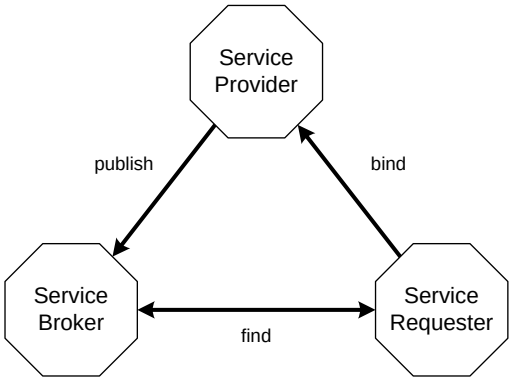


Figure 2.3: Roles in a service-oriented architecture [50]

tributed applications. As they may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration [170].

The service-oriented architecture (SOA) is an architectural style for building software systems based on services. Figure 2.3 depicts the three roles in SOA introduced by Burbeck [50]. A service provider *publishes* a service description to a service broker. A service requester can discover a service by querying the service broker (*find*). The broker passes one or several service descriptions back to the requester who can then *bind* to a service provider and subsequently interact with it. Furthermore, SOA is a set of architectural principles, patterns, and criteria, which address characteristics such as modularity, encapsulation, separation of concerns, reuse, composability, and single implementation [130].

Business Process Management largely benefited from the proliferation of SOA, since the important feature regarding it, is changing the focus from fine-grained, technology-oriented entities like database rows or Java objects, focusing instead on business-centric services with business-level transaction granularity [144].

2.2.1 Web Services

While SOA itself does not prescribe any particular technology for implementation, the web services platform architecture is the typical realization of it. Here, services are offered as *web services* [62]. According to the World Wide Web Consortium (W3C[2]) a web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format, specifically the Web Service Description Language (WSDL [56]). Other systems interact with the web service in a manner prescribed by its description using SOAP messages [44], typically conveyed using HTTP [111] with an XML [210] serialization. This serialization can be described using the XML Schema Definition (XSD [119]).

WSDL allows to define simple interaction scenarios with a web service involving one-way

[2] See `http://w3.org/`

or two-way message exchanges. The latter are carried out in a synchronous manner: The service requester sends a request to the web service and blocks until it receives a response from it. An example could be a stock information service delivering current stock values. In the case of asynchronous communication the delay between request and response would be too long so that blocking behavior is not acceptable. Here, polling or callback techniques come into play. Polling means that the requester regularly checks the status of the previous request. Callback can be applied if the requester is a web service itself. Here, the service invokes the requester as soon as the response is available.

Universal Description, Discovery and Integration (UDDI [211]) is a standard for web service brokers. In addition, there is a variety of web sites providing a list of available web services, such as `www.strikeiron.com` or `www.remotemethods.com`.

2.2.2 Web Service Orchestrations

Web service orchestrations lie at the intersection of SOA and BPM. The basic idea is to compose individual services to form a higher-value service. Early composition attempts led to techniques such as data scraping of web sites: information is scraped from the HTML representation of web sites in regular intervals and the aggregated information is displayed in a different format on another web site. This technique is extremely fragile, in regards of information consistency and communication interfaces.

Web service orchestrations are orchestrations, where the basic activities involve message exchanges with web services. The orchestration engine itself is again a web service. In a typical scenario a request is sent to the orchestration engine which instantiates the orchestration for processing that request. During execution several other web services are invoked by the orchestration engine in a predefined order. Finally, a response is sent back to the requester.

Imagine the auctioning service realized as web service orchestration. The seller acts as service requester and invokes the auctioning service. The orchestration engine instantiates the orchestration upon arrival of the order message. Next, the engine invokes a set of internal web services, e.g. triggering the creation of a web page for the offer and scheduling an auction. Then, it sends a response back to the seller giving him the confirmation that the auction was created.

Figure 2.4 illustrates an orchestration engine. Incoming requests are mapped to the corresponding process instance. Often, a new process instance is created upon a request, in other situations messages are routed to existing process instances. In the latter case, correlation information is included in the messages to determine which process instance it must be routed to. Process instances in turn can invoke other web services.

The most widely used standard for defining web service orchestrations is the Business Process Execution Language (BPEL [108]). It allows to implement long-running web service orchestrations including synchronous and asynchronous web service calls. BPEL will be discussed and assessed in detail in Chapter 3.

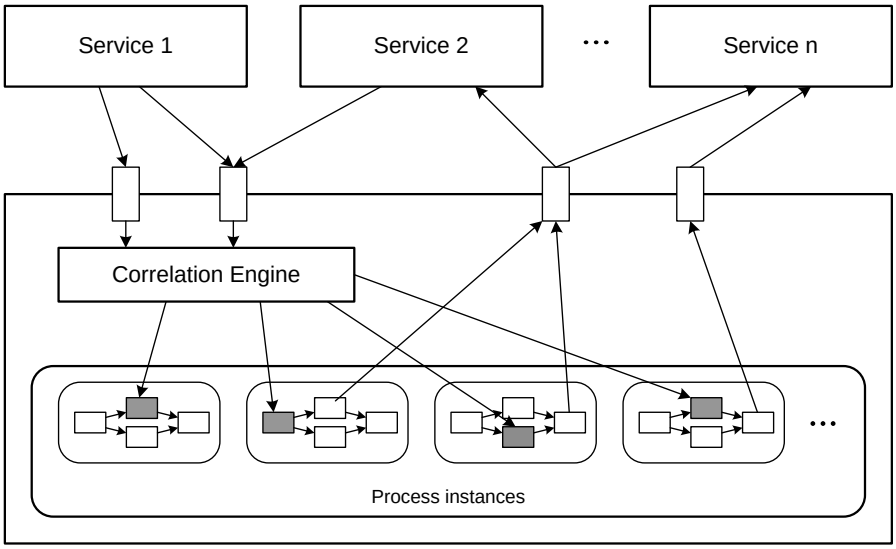


Figure 2.4: Web service orchestration engine, process instances and correlation

2.3 Choreography Languages

Just as process modeling is at the heart of Business Process Management, choreography modeling is at the heart of managing cross-organizational interaction. Choreography languages provide a standardized vocabulary and high-level modeling constructs for easily expressing typical real-world scenarios. This section enumerates the most prominent languages supporting the concepts of roles, interactions and behavioral dependencies. The list of languages includes Message Sequence Charts (MSC, [131]), the Business Process Modeling Notation (BPMN, [6]), UML Communication Diagrams [5], the Business Process Schema Specification (BPSS, [57]), the Web Services Business Process Execution Language (BPEL, [108]) and the Web Services Choreography Description Language (WS-CDL, [136]). An example from the auctioning domain will be presented for each language.

2.3.1 Message Sequence Charts (MSC)

Message Sequence Charts (MSC [131, 189]) originated in the telecommunication domain and were standardized by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T[3]). MSCs typically describe sample sequences of messages, as opposed to fully defining complex interaction behavior including concurrency, alternative branches and loops. MSCs are therefore suited for requirements engineering and the derivation of test cases. MSCs qualify as choreography language as multiple roles and interactions between them can be specified, including behavioral dependencies defining the ordering between outgoing and incoming messages of each role.

Figure 2.5 shows a message sequence chart illustrating an auctioning scenario. Roles are

[3] See `http://itu.int/ITU-T/`

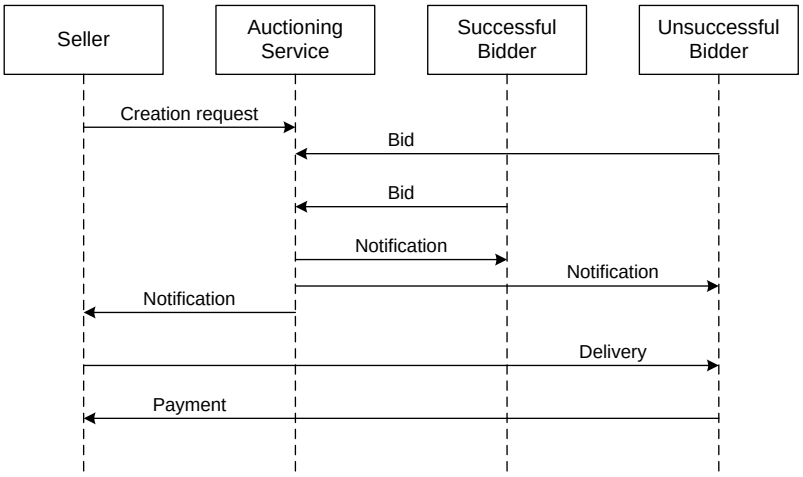


Figure 2.5: Message Sequence Chart (MSC) example

represented by rectangles with dashed lines (their *lifelines*). Interactions are represented by arrows. The behavioral dependencies between communication activities are given per role through the ordering of arrows starting and ending on the lifeline. The diagram illustrates only one conversation. In this case, exactly two bidders are involved and each bidder places exactly one bid. Due to their intuitive visual representation, MSCs are often used for discussing a particular message sequence. However, due to their absence of advanced control flow constructs they seldom represent complete choreographies.

2.3.2 Business Process Modeling Notation (BPMN)

The Business Process Modeling Notation (BPMN, [6]) is a graphical modeling language for intra- or inter-organizational business processes. BPMN was first released by the Business Process Management Initiative[4] in May 2004. Since then, BPMN evolved into a widely adopted language among business analysts. The fact that there are currently more than 50 BPMN tools available[5] underlines its popularity. After the merger of the Business Process Management Initiative with the Object Management Group (OMG)[6] a revised version was created in June 2005 and is now maintained as an OMG Final adopted specification [6]. The current version 1.2 [9] was released in February 2009.

Figure 2.6 shows the four main categories of modeling constructs in BPMN: swimlanes, connecting objects, flow objects, and artifacts. Swimlanes are used to separate organizational units involved in a collaboration or process. Pools are the top-level elements to structure a business process diagram (BPD) and represent roles or participants. Furthermore, a pool is structured into a hierarchy of lanes, representing business entities inside the organization of the pool. If a diagram contains only one pool with one lane, the graphical representation of the pool

[4] See `http://www.bpmi.org`
[5] See `http://www.bpmn.org/BPMNSupporters.htm`
[6] See `http://www.omg.org`

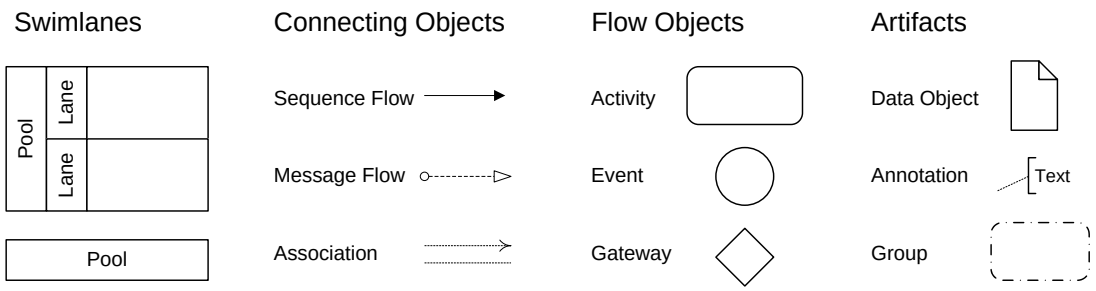


Figure 2.6: BPMN language overview

can be omitted.

BPMN specifies the concept of sequence flow (represented as arrow with solid lines) in order to express behavioral constraints between activities contained in the same pool. Therefore, a sequence flow is not allowed to cross the border of a pool. If multiple pools interact with each other the only way to model connections is via message flows (arrows with dashed lines). In case of an abstract process the message flows are directly connected to the pool. Message exchange within one pool is not allowed.

Associations can be used to connect artifacts with arbitrary flow objects in the diagram. In combination with data objects the directed variant is often used to indicate whether read or write access happens.

There are three categories of flow objects: activities are denoted as rounded rectangles, events as circles and gateways as diamonds. Each flow object is contained in one lane only and in doing so it is assigned to the organizational unit responsible for the execution.

A task is an elementary activity and the representation of an atomic piece of work to do. BPMN lists a number of different task types. Service tasks are used to call a piece of software in an application. Send and receive tasks have the same semantics as the corresponding message intermediate events. User tasks are pieces of work, which involve human interaction. Script tasks are similar to the service tasks, but can be executed directly by the process engine. Manual tasks do not involve the support of any process engine. Finally, reference tasks can reuse tasks defined elsewhere. Structured activities are called sub-processes. They can either contain a regular process or they are marked as ad-hoc and contain a set of tasks without any control dependency between them.

BPMN distinguishes between two different flavors of events. On the one hand, catching events are waiting for a trigger to fire. They can start a process as a reaction to a trigger or delay the flow of the process until the event occurs. On the other hand, throwing events can appear in the middle or at the end of a flow. In contrast to catching events, they do not wait for a trigger, but have a result.

Behavioral dependencies in BPMN are not only defined through sequence flow. In addition, different kinds of gateways allow the specification of advanced branching behavior and concurrency in business processes. Artifacts provide additional information and do not directly affect the behavior of a process. Data objects represent data relevant for the process. Associations indicate where the data is created, read or modified.

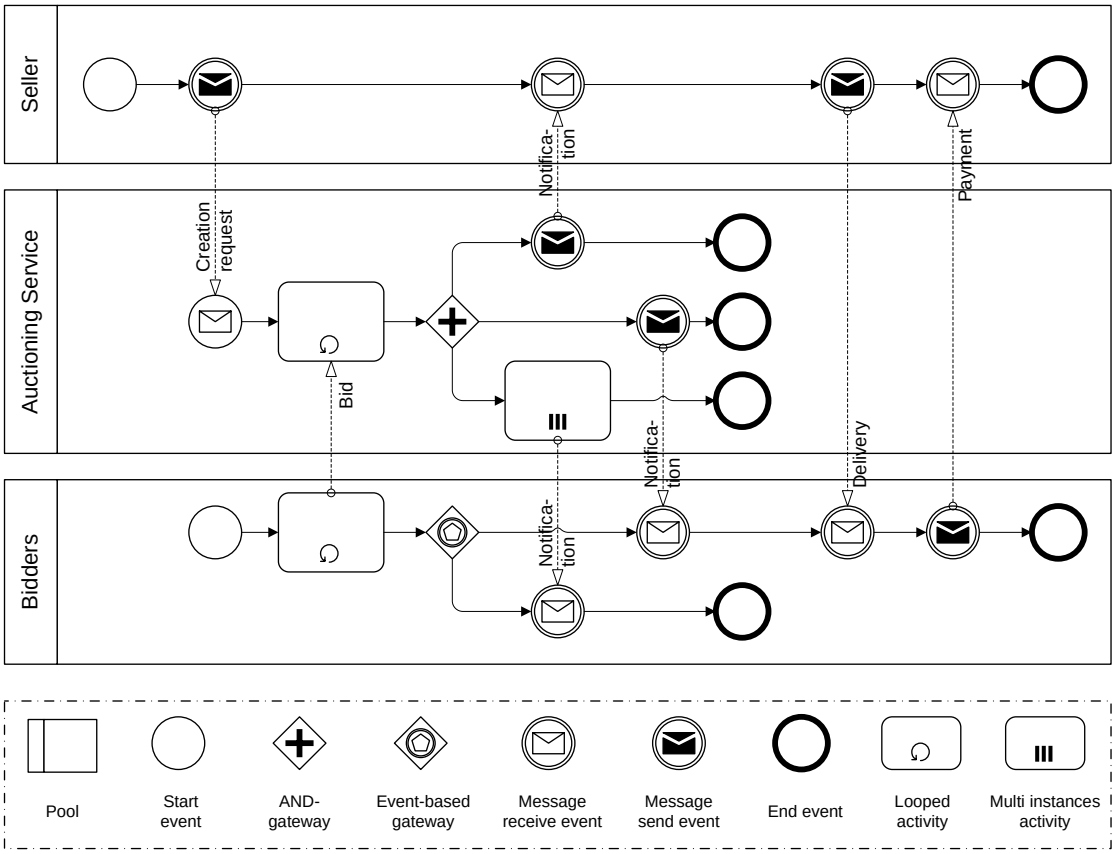


Figure 2.7: Business Process Modeling Notation (BPMN) example

Especially due to the concepts of pools and message flow, BPMN can be used for choreography modeling. The behavioral constraints between the different interactions are implicitly given through the sequence flow dependencies and gateways relating the communication activities.

Figure 2.7 illustrates an auctioning example with three roles and a number of interactions involved. It shows how advanced control flow constructs such as loops, multiple-instance activities and gateways can be combined with message flow between roles.

The suitability of BPMN for modeling typical business process scenarios was studied in [216] and the actual use of BPMN language constructs in practice was investigated in [223].

2.3.3 Business Process Schema Specification standard (BPSS)

The Business Process Schema Specification standard (BPSS [99]) is used to describe how two or more roles interact in an ebXML scenario. ebXML is an initiative by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT)[7] that was introduced as alternative to heavyweight EDI-solutions for small and mid-sized companies.

[7] See `http://www.unece.org/cefact/`

BPSS is closely related to the UN/CEFACT Modeling Methodology (UMM, [7]) which mainly describes the different steps to specify choreographies in a technology-independent manner. UMM also provides a meta-model for choreographies, including the business transactional view, the business service view and business collaboration view. Business transactions serve as basic building blocks, each involving a request-response interaction plus additional business signals to synchronize the state of the two business partners involved. It is suggested to describe these bi-lateral transactions using UML 1.4 Activity Diagrams [1]. Transactions are composed into a business collaboration protocol, the choreography. Again, it is suggested to describe the choreography using Activity Diagrams or BPMN, however, an integration with these languages is not provided. UMM's business service view finally specifies the services and operations (or messages) participants must support in order to implement a role.

The specification of business document flow is one of the main strengths of BPSS choreographies. Another ebXML standard, namely the Core Components Technical Specification (CCTS [61]), introduces a means to specify business documents in a technology-agnostic way. Information interoperability is achieved by agreeing on business semantics for the information. The so called "core components" represent generic business information that can then be used in the form of so called "business information entities" in different business contexts. Two different

```
<BusinessCollaboration name="Delivery and Payment">
  <Role name="seller" nameID="s"/>
  <Role name="bidder" nameID="b"/>
  <TimeToPerform duration="P5D"/>
  <Start><ToLink toBusinessStateRef="Delivery"/></Start>
  <BusinessTransactionActivity businessTransactionRef="Delivery">
    <Performs currentRoleRef="s" performsRoleRef="CCinitiator1"/>
    <Performs currentRoleRef="b" performsRoleRef="CCresponder1"/>
  <BusinessTransactionActivity/>
  <BusinessTransactionActivity businessTransactionRef="Payment">
    <Performs currentRoleRef="b" performsRoleRef="CCinitiator1"/>
    <Performs currentRoleRef="s" performsRoleRef="CCresponder1"/>
  <BusinessTransactionActivity/>
  <Transition>
    <FromLink fromBusinessStateRef="Delivery"/>
    <ToLink toBusinessStateRef="Payment"/>
  </Transition>
  <Decision>
    <FromLink fromBusinessStateRef="Payment"/>
    <ToLink toBusinessStateRef="Success">
      <ConditionExpression expression="Success"/>
    </ToLink>
    <ToLink toBusinessStateRef="Failure">
      <ConditionExpression expression="Failure"/>
    </ToLink>
  </Decision>
  <Success nameID="Success"/>
  <Failure nameID="Failure"/>
</BusinessCollaboration>
```

Listing 1: Business Process Schema Specification example

contexts can be given e.g. through different countries or legislations. While a company in Australia can be uniquely identified by an ABN (Australian Business Number), a German company can be identified by its *Handelsregisternummer*. However, in both cases we deal with the core component *company*. Business information entities can be encoded using different syntaxes, e.g. XML or United Nations/EDI for Administration, Commerce and Transport (UN/EDIFACT).

Listing 1 shows a BPSS example. The choreography contains two interactions, namely delivery and payment. The behavioral dependencies are defined using `Start`, `Transition`, `Fork`, `Join` and `Decision` primitives. Successful and unsuccessful completion of conversations can be distinguished using the `Success` and `Failure` constructs.

2.3.4 UML Communication Diagrams

UML Communication Diagrams are part of the Unified Modeling Language (UML) 2.0 standard [5]. Communication diagrams can be used for modeling interactions among distributed components without exposing their internal structure. They assume that the order of message sending and message receipt are the same or are irrelevant to the modeler. Therefore, message send and message receipt are modeled as atomic unit.

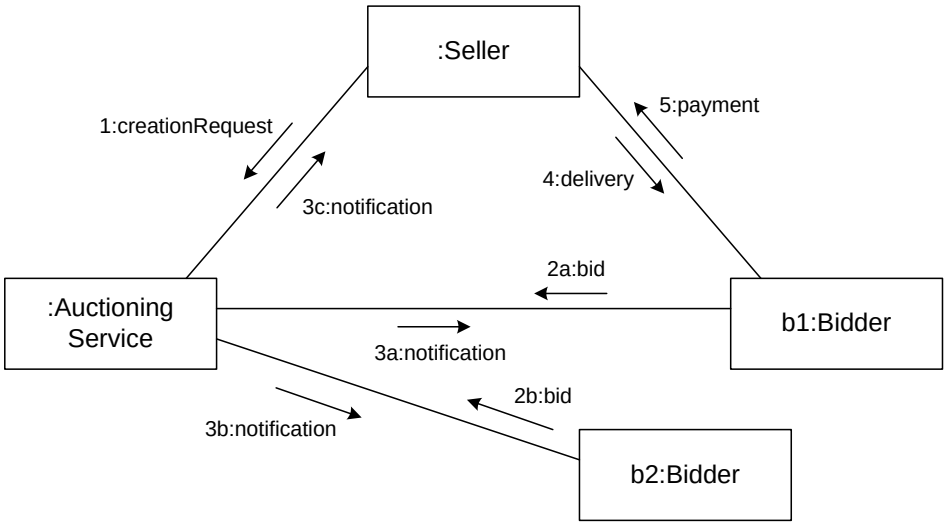


Figure 2.8: UML Communication Diagram example

All behavioral dependencies between interactions are specified using so called sequence expressions. These textual expressions allow sequences of messages, hierarchy, parallelism, alternative branches, iterations and multiple instances interactions. Figure 2.8 shows a sample communication diagram from the auctioning domain. The rectangles, called lifelines, represent participants of a particular role. One participant of role seller, one of role auctioning service and two of role bidder are included in the diagram. Therefore, an exemplary conversation is illustrated rather than a complete choreography.

The sequence expressions specify that first the creation request is sent from the seller to the auctioning service, then two bids are placed concurrently, before three notifications are sent

concurrently. Finally, the delivery and payment interactions happen. The numbers indicate a sequential order and “a” .. “c” indicate parallelism.

2.3.5 Business Process Execution Language (BPEL)

The Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short) was submitted for standardization to the Web Services BPEL Technical Committee of the Organization for the Advancement of Structured Information Standards (OASIS)[8] in 2002. Version 1.1 was finalized in May 2003 [25]. The current version 2.0 [108] was renamed to Web Service Business Process Execution Language (WS-BPEL) to conform to other so called WS-* specifications.

BPEL has its origins in two process implementation languages developed by the main contributors to the specification, Microsoft and IBM. On the one hand XLANG [203] was developed by Microsoft, on the other hand the Web Service Flow Language (WSFL) [147] comes from IBM. Both languages are based upon the Web Services Description Language (WSDL) [56]. Nevertheless, for the description of processes they use fundamentally different paradigms. While XLANG defines the process with structured nested statements, WSFL uses a graph-structured paradigm for process modeling. This difference can still be seen in their successor. BPEL is a mainly block-structured language [167, 10] and activities can be nested in arbitrary ways like in XLANG. Furthermore, it provides a mechanism to formulate graph-structured process models.

BPEL distinguishes two types of processes. On the one hand, abstract processes model the interactions between business partners via web services. The process model does not need to be complete, because it is not intended for execution. On the other hand, executable processes are not allowed to omit any operational details. They can be executed in a process engine.

The main ingredients of an executable BPEL process in order to provide and consume web services are partner links and their type definitions, variables, handlers and activities. Regarding its use to specify processes, it is similar to imperative programming languages, where the steps to be done are explicitly formulated. For example, the structured activities define the way how basic activities are executed. Basic activities model the atomic pieces of work to do.

The partner links and partner link types are a lightweight mechanism to link the communication with business partners on a conceptual layer, with the underlying web service descriptions. In contrast to partner links, the types are defined inside the actual WSDL file. Each process must come with a set of related WSDL files, including the used partner link types and further data description of the services the process should call or provide.

BPEL distinguishes basic and structured activities. Basic activities mainly include communication activities for message sending and message receipt (invoke, receive, reply, pick, ...) and internal activities e.g. for data transformations and raising exceptions. Structured activities are used to describe the control flow, allowing for concurrency, alternatives, loops and even multiple-instances. An exhaustive assessment of BPEL along the Workflow Patterns [17] can be found in [217].

Listing 2 shows a BPEL example describing the behavior of the auctioning service. A process instance is created upon receipt of an auction creation request message. After that, bids are

[8] See `http://www.oasis-open.org`

```
<process name="auctioningService" ...>
  <sequence>
    <receive operation="requestAuctionCreation" createInstance="yes" .../>
    <repeatUntil><condition>...</condition>
      <pick>
        <onMessage ...>...</onMessage>
        <onAlarm><until>...</until>...</onAlarm>
      </pick>
    </repeatUntil>
    <flow>
      <invoke .../>
      <invoke .../>
      <forEach parallel="yes" ...><scope>
        <invoke .../>
      </scope></forEach>
    </flow>
  </sequence>
</process>
```

Listing 2: BPEL example

received repeatedly until a timeout occurs (realized through `onAlarm`). Notifications are sent out concurrently within the `flow` structure.

BPEL^{light} [162] has been introduced as extension of BPEL to decouple BPEL and its technical details. In essence, it loosens BPEL's strong dependency on WSDL.

BPEL is a typical orchestration language, mainly focusing on the activities within the service that is implemented by means of a BPEL process. However, abstract BPEL also allows to focus on the communication activities and behavioral dependencies between them. Although multiple roles can be mentioned in a BPEL process through the notion of partner links, BPEL does not qualify as choreography language as it concentrates on one role only and is meant to be executed by a central orchestrator.

In contrast to this, WSFL, the predecessor of BPEL, can also be considered a choreography language due to its distinction between *flow models* and a *global model*. While the flow models define the behavioral dependencies between activities, the global model provides the interconnection between the different flow models.

2.3.6 Web Services Choreography Description Language (WS-CDL)

The Web Services Choreography Description Language (WS-CDL [136]) is probably the best known choreography language in the web services world at the moment. It was released by the World Wide Web Consortium (W3C)[9] as Candidate Recommendation in November 2005. This indicates that the standard is expected to be stable and that implementation is encouraged.

WS-CDL supersedes its two predecessors Web Services Choreography Interface (WSCI [28]) and Web Service Conversation Language (WSCL [33]). WSCI has considerable overlap with the Business Process Modeling Language (BPML [27]), proposed as orchestration lan-

[9] See `http://www.w3.org/`

guage by the Business Process Management Initiative. WSCL has its roots in Hewlett-Packard's Conversation Definition Language (CDL [146]).

WS-CDL is an XML-based language focusing on interactions and their relationships. Interactions are bi-lateral (between two web services) and involve either one message (request-only or response-only) or two messages (request-response). Each interaction takes place via a channel instance identifying the responding service. An important feature of WS-CDL is the possibility to pass channel instances from one service to another in an interaction. The structure of other information that can be passed is specified using information types. Role types define what behaviors (WSDL interfaces in the default case) have to be implemented by a corresponding service. Relationship types are pairs of role types, services of which can directly interact with each other. Correlation of interactions is addressed using identity tokens that are to be included in messages. Interactions can be composed to activities using a range of control flow constructs. The comparison of WS-CDL and π-calculus in [83] gives a more detailed insight into the semantics of the control flow constructs of WS-CDL.

WS-CDL does not come with a graphical representation. As the name already indicates, WS-CDL is closely tied to web services architectures. Especially a tight integration with WSDL is given. Relationships to other web services standards such as WS-Security [4] are mentioned but not given much attention to. There have been proposals for generating BPEL abstract processes out of WS-CDL choreographies [159]. The tool pi4soa[10] implements generation algorithms. However, it remains unclear how essential constructs such as blocking work units can be mapped to BPEL. The WS-CDL specification states that state machines could be generated for every participant in a WS-CDL choreography.

```
<choreography>
  <sequence>
    <interaction name="auctionCreation" channelVariable="tns:broker-channel"
       operation="requestAuctionCreation" initiate="true">
      <participate relationshipType="tns:SellerASRel"
          fromRole="tns:Seller" toRole="tns:Broker"/>
      <exchange name="request" informationType="tns:creationReq"
          action="request">
        <send/>
        <receive variable="..." />
      </exchange>
      <exchange name="response"...>...</exchange>
      <timeout time-to-complete="..."/>
    </interaction>
    <workunit repeat="...">
      <interaction name="bid" ... operation="placeBid">...</interaction>
    </workunit>
  </sequence>
</choreography>
```

Listing 3: Sample interaction in WS-CDL

Listing 3 shows a WS-CDL example for our auctioning scenario. The auction creation

[10] See `http://sourceforge.net/projects/pi4soa/`

interaction initiates a conversation. After having completed this interaction, a workunit marked as repetition is enabled. The workunit contains the bid interaction between bidders and the auctioning service.

2.3.7 Other Choreography Languages

Several industry initiatives have worked towards domain-specific choreography models in order to enable easier integration between different companies of that domain. Examples for such initiatives are RosettaNet[11] for the supply chain domain, SWIFTNet[12] (Society for Worldwide Interbank Financial Transfer) for financial services or HL7[13] (Health Level Seven) for health care services. Due to a lack of choreography modeling languages available, these initiatives have mostly resorted to textual descriptions of the choreographies. Rather adhoc-notations where used for illustration. However, both the textual descriptions and the illustrations have many ambiguities, allowing for different interpretations. When it comes to execution, the current way is to map the proprietary notations to BPEL by making heavy assumptions about the concrete semantics. Such a mapping is currently available for RosettaNet only and presented in [138].

WSDL 2.0 Message Exchange Patterns [163] offer means to specify the order of messages an operation expects and sends back. This ordering is specified using text and not a designated process description language. It is possible to specify node types, which enables describing multi-lateral interactions from the view of one participant. However, the specification is limited to one operation and thus does not cover a whole choreography spanning multiple operations.

The Service Component Architecture (SCA) provides a model for building applications based on SOA [109, 165]. Implementations of services are wrapped in components. Each component provides a set of services and requires a set of interfaces. The required services have to be wired with provided services to form a valid SCA composite. SCA itself does not provide a specific language for the implementation of components. SCA supports BPEL as a possible implementation language [166].

Seel et al. [197] present a requirements framework for inter-organizational business process models. A distinction is made between interaction points for collaborating employees and departments and interaction points for information systems. Corresponding extensions to Event-driven Process Chains (EPC [137]) are introduced.

UML-RT [198] introduces the notions of capsules, ports, connectors, protocol roles and protocols. Capsules interacts with an environment through ports and they are interconnected using connectors. A port realizes a protocol role, which in turn collective form a protocol.

Most languages mentioned above are procedural languages, where the approach is to explicitly enumerate all interactions possible in a certain situation. As an alternative to these procedural languages there are also approaches for a declarative style of modeling [14, 173]. Declarative in this context means that all constraints are enumerated that apply for a set of interactions. That way, an "empty" choreography would mean that all interaction sequences are allowed, while adding constraints limits the number of allowed sequences. In procedural languages, an

[11] See `http://www.rosettanet.org/`
[12] See `http://www.swift.com/`
[13] See `http://www.hl7.org/`

"empty" choreography would mean that no interaction is allowed and adding constructs enlarges the set of possible interaction sequences. The goal of declarative languages is to avoid over-specification, a common phenomenon that can be observed whenever procedural languages are used. An overview of declarative workflow models is given in [174].

Quartel et al. proposed the Architecture Description Language ISDL and discussed its application for service choreography modeling in [183].

2.4 Choreography Formalisms

Formal models allow the unambiguous definition of behavioral dependencies in orchestrations and choreographies. They provide an abstract syntax with precise execution semantics that also facilitates automatic reasoning on properties of the choreographies. The abstract syntax and the semantic definitions of the different formalisms are typically so compact that algorithms can be defined in a compact way and formal proofs are feasible.

This section lists the most prominent formal approaches used in the choreography space, including communicating finite state machines, conversation models, open nets and π-calculus.

2.4.1 Communicating Finite State Machines

Finite state machines (also called finite state automata) describe the relationship between states of a system and the transitions allowed between these states. The number of states is finite and they can therefore be enumerated and represented individually in a diagram. Finite state machines are a widely used formalism for describing the behavior of systems [129].

Finite state machines (FSMs) have a distinguished initial state, i.e. the state the system is in when being started, and a set of final states, i.e. states which it would be allowed for the system to stop in. Transitions are labeled and indicate what states can be reached next.

Definition 2.1 (Finite State Machine) A *finite state machine (FSM)* is a tuple $(A, S, s_0, final, \delta)$ where A is an alphabet, S is the set of states, $s_0 \in S$ is the initial state, $final \subseteq S$ is the set of final states and $\delta \subseteq S \times A \times S$ is the transition relation. □

Reachability of states can be derived from the transition relation. A state s is reachable, if there is a path of transitions from the initial state to s. On top of that, the introduction of final states allows the identification of deadlocks.

Definition 2.2 (Deadlock) Given a finite state machine $(A, S, s_0, final, \delta)$. A *deadlock* is a reachable state $s \in S$ without outgoing transitions, i.e. $\nexists a, s' \ ((s, a, s') \in \delta)$, and $s \notin final$. □

Communicating FSMs [154, 64, 45, 172] are a formal model for interconnecting several independent FSMs through message exchanges. In this context, a FSM can do send or receive actions or alternatively do internal steps that cannot be observed by the environment. The alphabet used by the communicating FSMs is $(\{!, ?\} \times M) \cup \{\tau\}$ where M is the set of messages and ! and ? indicate sending or receiving. $(s, !m, s')$ denotes a message send action from role $send(m)$ to role $recv(m)$ and $(s, ?m, s')$ a message receive action. (s, τ, s') denotes an internal, i.e. silent, transition.

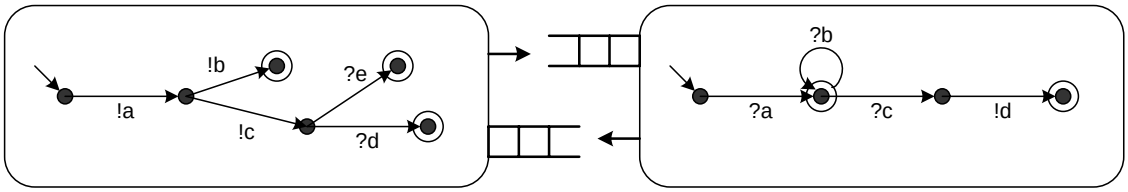


Figure 2.9: Communicating Finite State Machine example

Figure 2.9 illustrates communicating finite state machines, where the states are depicted by circles and the transitions by arrows. The initial state is targeted by an arrow without source state and final states are denoted by double-circles. Each FSM has a queue for incoming messages.

Definition 2.3 (Composition of communicating FSMs) Given a set of communicating FSMs $R = \{r_1, \ldots, r_n\}$. Their composition is a (2n)-tuple of the form $(Q_1, s_1, \ldots, Q_n, s_n)$ where Q_i denotes the queue contents of role r_i and s_i denotes the state of r_i. □

The inclusion of queues allows the representation of both synchronous and asynchronous communication. In the case of synchronous communication, the message is put into the input queue and read from it in a single step. In the case of asynchronous communication, a message is placed into the queue and can be read as soon as the corresponding FSM is ready to receive it.

Different flavors of asynchronous communication are possible: the buffer size might be either finite or infinite and the order of message delivery might be First-In-First-Out (FIFO) or any other ordering scheme. There might be one queue per FSM or one queue per message type. In case of synchronous communication or asynchronous communication with bounded queues, the behavior of a composition of communicating FSMs can again be described as FSM.

2.4.2 Conversation Models

While communicating FSMs define behavioral dependencies on a per-role level, conversation models are a formalism where behavioral dependencies between interactions are defined globally [113]. Conversation models also build on FSMs. The set of messages is again denoted as M. $send(M)$ denotes the sender of m and $recv(m)$ the receiver of m.

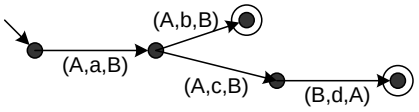


Figure 2.10: Conversation model example

Figure 2.10 shows a conversation model where an interaction (A, a, B) denotes that a message a is sent from A to B. Conversation models can both be applied in synchronous and asynchronous settings. While in synchronous settings the semantics of the ordering constraints is intuitive, the constraints only apply to the ordering of send activities in the presence of asynchronous communication. The order in which messages must be received are not specified in this case.

2.4.3 Open Nets

Petri nets were introduced by Carl Adam Petri in [175] and are now widely used in the field of Business Process Management [18, 19]. In contrast to the formalisms from the previous sections, Petri nets provide direct support for expressing concurrency.

Petri nets are bipartite directed graphs consisting of places and transitions that are connected through a flow relation. Places can contain tokens and the token flow determines the firing sequences of the transitions. Upon firing of a transition, some tokens from input places are removed and some tokens are placed on output places.

The most basic form of Petri nets are *place/transition nets*. Here, a transition is enabled, i.e. ready to fire, if there is at least one token in each input place of that transition. Firing of this transition results in removing one token from each input place and placing one token in each output place [188].

Definition 2.4 (Petri Net) A *Petri net* is a tuple (P, T, F, m_0) where P and T are disjoint sets of places and transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation connecting places and transitions and $m_0 : P \to \mathbb{N}$ is the initial marking. □

As auxiliary notation we denote the input and output places of a transition t with $\bullet t$ and $t\bullet$, where $\bullet t := \{p \in P \mid p\ F\ t\}$ and $t\bullet := \{p \in P \mid t\ F\ p\}$. $\bullet p$ and $p\bullet$ denote the set of transitions that share p as output and input places, respectively, i.e. $\bullet p := \{t \in T \mid t\ F\ p\}$ and $p\bullet := \{t \in T \mid p\ F\ t\}$.

We are going to use an abbreviated notation for markings. $[p_1, p_2]$ denotes that one token resides on place p_1, one on p_2 and all other places are empty. $[p_1, p_1]$ denotes that two tokens reside on place p_1.

The execution semantics of Petri nets is given by their token flow. Progressing from one marking to another marking happens through firing of transitions. Firing results in removing tokens from input places and adding tokens to output places of the transition.

Definition 2.5 (Enablement and Firing) Let (P, T, F, m) be a Petri net. A transition $t \in T$ is enabled iff $m(p) > 0$ for all $p \in \bullet t$. The reached marking after firing of t is m', where $m'(p) = m(p) - 1$ for all $p \in \bullet t \setminus t\bullet$, $m'(p) = m(p) + 1$ for all $p \in t\bullet \setminus \bullet t$ and else $m'(p) = m(p)$. We denote this as $(P, T, F, m) \xrightarrow{t} (P, T, F, m')$ or $m \xrightarrow{t} m'$ for short. □

When $m \xrightarrow{t} m'$ we say that m' is reachable from m by firing transition t. Reachability can be canonically extended to transition sequences.

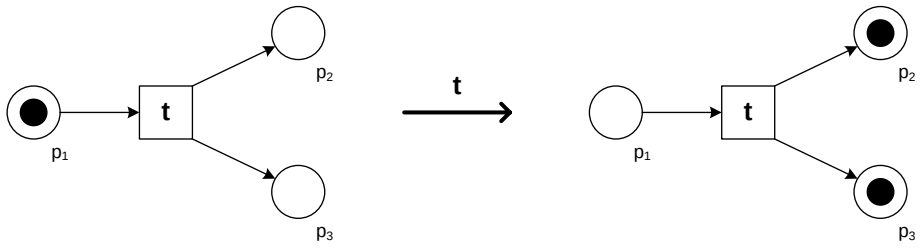


Figure 2.11: A Petri net before and after firing of transition t

Figure 2.11 depicts two Petri nets (P, T, F, m_1) and (P, T, F, m_2), where $P = \{p_1, p_2, p_3\}$, $T = \{t\}$, $F = \{(p_1, t), (t, p_2), (t, p_3)\}$, $m_1 = [p_1]$ and $m_2 = [p_2, p_3]$. The places are graphically represented as circles, the transitions as rectangles and the flow relation as arrows. The marking is represented by small filled circles, the tokens, contained in the places. In case of m_1, transition t is enabled as all input places $\bullet t = \{p_1\}$ contain tokens. Firing of t leads to marking m_2, where the token on p_1 was removed and two tokens were produced on places p_2 and p_3.

Figure 2.12 illustrates a second example from the auctioning domain. It shows the behavior of a seller. Activities are represented by transitions. By letting the tokens flow through the net, we see that *Set up auction* is always followed by *Receive confirmation* and the payment and delivery activities can happen concurrently. The marking that is finally reached is $[s_9]$.

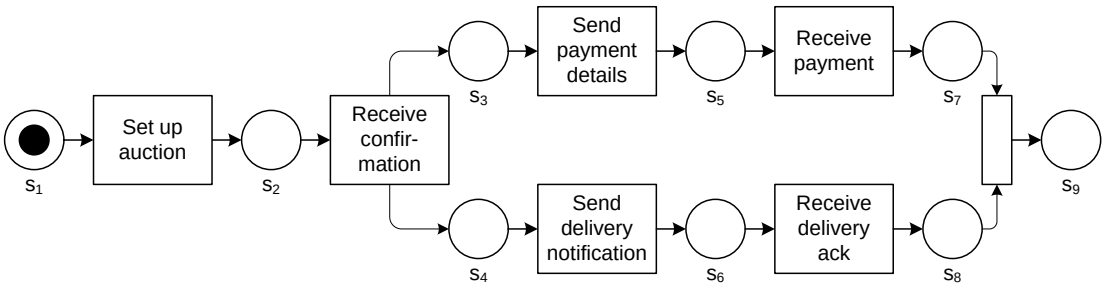


Figure 2.12: The seller represented as Petri net

The example shows that Petri nets natively support the notion of concurrency. Therefore, those cases where different actions can happen in any order (or might even overlap in execution), can be represented in a much more compact way in Petri nets than it is the case for finite state machines. Each activity related to payment and delivery is only represented once in the Petri net.

The Petri nets presented so far do not include the notion of communication. Therefore, we use *open nets* [151] where internal control flow and communication via message exchanges are distinguished through two types of places: internal places and communication places.

Definition 2.6 (Open Net) An *open net* is a tuple $(P, P_C, T, F, m_0, final)$ where (P, T, F, m_0) is a Petri net, $P_C \subseteq P$ is the set of communication places (all other places $P \setminus P_C$ are internal places) and $final \subseteq P \rightarrow \mathbb{N}$ is the set of valid final markings. For all communication places $p_C \in P_C$ it must hold $\bullet p_C = \emptyset \ \vee \ p_C \bullet = \emptyset$ and $m_0(p_C) = 0$. □

The enabling and firing rules of Petri nets apply to open nets unchanged. The additional rules for communication places ensure that there must not be any communication place that is "send" place and "receive" place at the same time and that all communication places are empty in the initial marking. In analogy to FSMs, the introduction of valid final markings allows the distinction of desired markings and deadlocks.

Definition 2.7 (Deadlock) A marking m is a deadlock iff no other marking is reachable from m and $m \notin final$. □

Figure 2.13 depicts an open net describing the seller. In addition to the places and transitions

from Figure 2.12, communication places were added. This is visualized by the dashed rectangle.

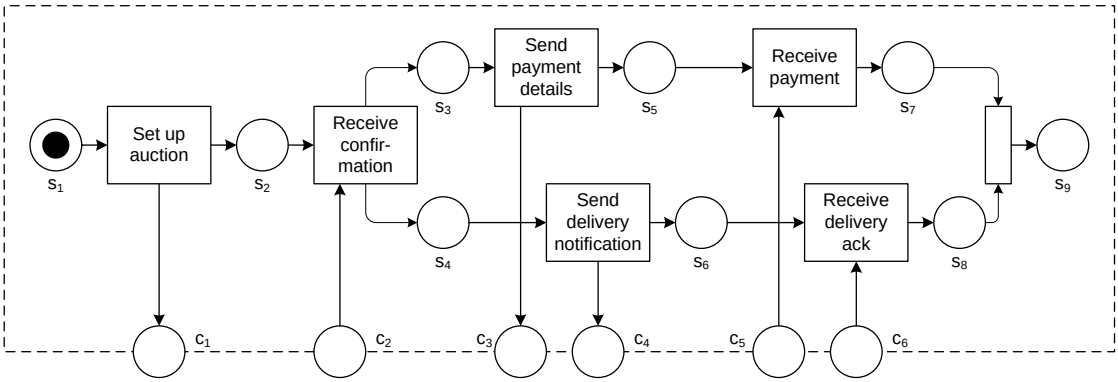


Figure 2.13: The seller represented as open net

An open net defines the internal behavior and interaction behavior of one role. The behavior of two or more interacting roles is defined through the composition of the open nets to a new open net. The basic idea of this composition is to merge corresponding communication places. That way tokens produced on a communication place by one role can be consumed by another role.

Definition 2.8 (Closed Composition of Open Nets) Two open nets $O_1 = (P_1, P_{C1}, T_1, F_1, m_{01}, final_1)$ and $O_2 = (P_2, P_{C2}, T_2, F_2, m_{02}, final_2)$ are composable iff $P_1 \cap P_2 = P_{C1} \cap P_{C2}$ and $T_1 \cap T_2 = \emptyset$. The *closed composition* of O_1 and O_2, denoted as $O_1 \oplus O_2$, is $(P_1 \cup P_2, (P_{C1} \cup P_{C2}) \setminus (P_{C1} \cap P_{C2}), T_1 \cup T_2, F_1 \cup F_2, m_{01} \oplus m_{02}, \{f_1 \oplus f_2 \mid f_1 \in final_1 \wedge f_2 \in final_2\})$. $\square$

By abuse of notion, the operator $m_1 \oplus m_2$ also denotes the composition of two markings of composable open nets, where $(m_1 \oplus m_2)(p) = m_1(p)$ iff $p \in P_1$, and $(m_1 \oplus m_2)(p) = m_2(p)$ otherwise. Figure 2.13 depicts the closed composition of two open nets, one taken from Figure 2.13 and the second one representing parts of an auctioning service. Places c_1 and c_2 that were previously communication places are now internal places. Places c_3 through c_6 remain communication places.

The composition is closed, as communication places "disappear" during composition. The underlying assumption is that no other role can access the communication place after the composition. This might be intuitive in the case of sending (during composition it is decided that a message is targeted at a specific role). However, for receive places the assumption that the place is "locked" and cannot be filled by any other role might be counterintuitive. Furthermore, the composability of a set of open nets might not be given if multiple open nets send on (or receive from) the same communication place.

In an open composition all communication places from the original nets remain communication places. The idea of open composition and a comparison to closed composition can also be found in [78].

Definition 2.9 (Open Composition of Open Nets) The *open composition* of two composable nets O_1 and O_2, denoted as $O_1 \oplus_{open} O_2$, is $(P_1 \cup P_2, P_{C1} \cup P_{C2}, T_1 \cup T_2, F_1 \cup F_2, m_{01} \oplus$

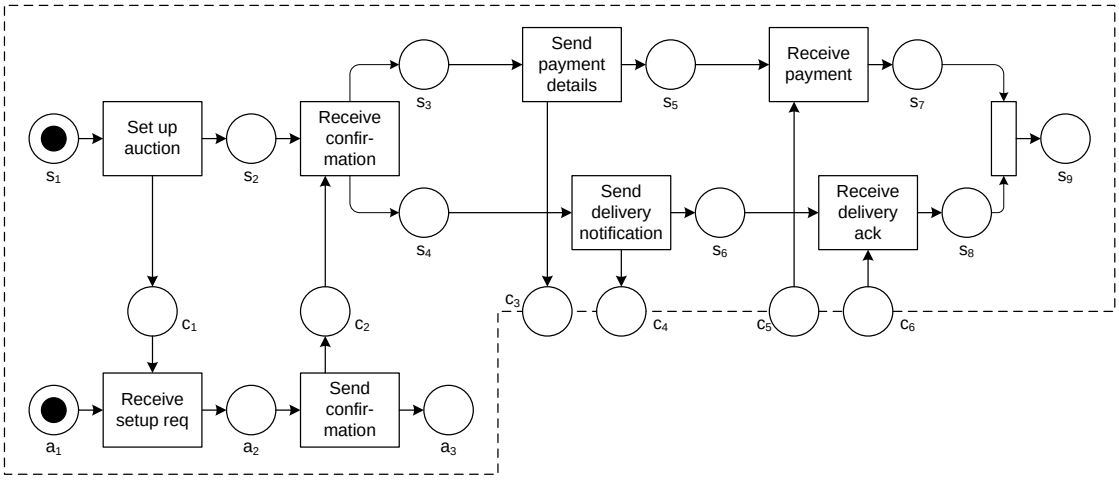


Figure 2.14: Closed composition of open nets

$m_{02}, \{f_1 \oplus f_2 \mid f_1 \in final_1 \wedge f_2 \in final_2\})$. □

2.4.4 Pi-calculus

The π-calculus is a process algebra for mobile systems, which sees interacting processes as central concept. All communication takes place via names. A precondition for two processes to communicate via such a name is that both processes know this name. While some names might be known to all processes from the very start of a conversation, there is the possibility to propagate knowledge about names at runtime. This happens by passing names to other processes. This phenomenon is called *link passing mobility*. While the scope of a name might be restricted to one process at the beginning of the conversation, it is extruded as soon as it is passed to the other process.

$$\begin{aligned} S &= s(x).\overline{x}.\mathbf{0} \\ A &= \overline{s}\langle b\rangle.\mathbf{0} \\ B &= b.\mathbf{0} \\ SYS &= S \mid A \mid B \end{aligned}$$

The above example illustrates some of the concepts of π-calculus. We find a system of three interacting processes S (a seller), A (an auctioning service) and B (a buyer). This example sets in after an auction has finished and B was selected as the successful bidder by A. At this point in time, S does not know yet who is going to be the buyer for the goods offered. Therefore, A has to pass on the reference of B on to S. A knows the names s and b and sends b over s. S also knows s and receives a name over this channel. x is a place holder for the name to be received. Upon receipt, x is substituted by the received name—in our case b. Therefore, S evolves to $S' = \overline{b}.\mathbf{0}$. Finally, S' and B can communicate as b is known to both S' and B. We call s and b the free names of A. S has only one *free name*, namely s, and one *bound name*: x.

π-calculus provides direct support for expressing sequences, alternative branching (choices) and concurrent branching. Synchronization is exclusively done through receive actions. Recursion is supported as well.

Another important feature of π-calculus is the creation of fresh names. Fresh means that the newly created name does not appear as free name anywhere else in the system. It is possible to create an infinite number of names by combining name creation with recursion. Name creation is especially useful for formalizing synchronous request/response-cycles or more complex interaction scenarios involving correlation. Before sending the request, the requesting party creates a fresh name and passes this name on to the other party, who in turn uses this name as response channel.

The syntax of π-calculus is defined as follows:

$$\begin{aligned} P &::= M \mid P|P' \mid (\nu z)P \mid !P \\ M &::= \mathbf{0} \mid \pi.P \mid M + M' \\ \pi &::= \overline{x}\langle y\rangle \mid \overline{x} \mid x(y) \mid x \mid \tau \end{aligned}$$

Concurrent execution is denoted as $P|P'$, the restriction of the scope of z to P as $(\nu z)P$ and an infinite number of concurrent copies of P as $!P$. Inaction of a process is denoted as $\mathbf{0}$, a non-deterministic choice between M and M' as $M+M'$, sending y over x as $\overline{x}\langle y\rangle$, sending an empty message over x as $\overline{x}$ and receiving an empty message over x as x. The prefix $x(y)$ receives a name over x and continues as P with y replaced by the received name. τ is the unobservable action. Communication between two processes can take place in the case of matching send- and receive-prefixes.

There are two definitions for the execution semantics of a π-process. The reduction semantics prescribes structural rules how to alter π-processes upon interaction or internal activities. Alternatively, transition system semantics is available. More details on the semantics of π-calculus can be found in [160].

Puhlmann has investigated the applicability of π-calculus to the area of Business Process Management and Service-oriented Architectures in [178]. He concludes that it provides an intuitive representation of many recurring scenarios. As part of that he provides a formalization of all Workflow Patterns [17] in [179], showing that even advanced workflow patterns such as multiple instances with a-priori runtime knowledge can be expressed using π-calculus. Only few tools are available for π-calculus, e.g. the Advanced Bisimulation Checker [47] and the Mobility Workbench [209].

2.4.5 Other Formalisms

The previous section has presented a number of prominent formalisms in the choreography space. However, there has been much work presenting alternative approaches. Su et al. provide a survey on different formalisms in [200]. It categorizes the formalisms along the underlying communication model and whether ordering constraints are provided on a per-role level or on a global level.

Reo is a formalism based on mobile channels [26]. Channels can be composed to more complex connectors, which in turn allow the specification of choreographies. Therefore, Reo

focuses on interaction behavior between software components while hiding all internal behavior of components. The SENSORIA reference modeling language (SRML, [134]) also allows specification of interaction protocols between services. So called modules with provides- and requires-interfaces are connected through wires. The behavioral dependencies between interactions are given as doubly-labeled transition system.

Carbone et al. provide a formal model following the interaction modeling style in [54], which was strongly inspired by WS-CDL. Berardi et al. present *Colombo*, a formal model that also includes data handling in service compositions [40]. Busi et al. present their own process algebra for describing synchronous interconnection models [52]. Qiu et al. present a process algebra for interconnection models [182], including the concept of a dominant role for choices and loops.

As Petri nets cannot properly represent value passing and the distinction between different process instances, colored Petri nets were introduced [133]. The basic idea is that each token carries a value, e.g. representing the contents of a business document or a message. The enablement of transitions can be restricted in such a way that a certain combination of values is required. In addition, functional dependencies between input values and output values of a transition are defined.

Engels et al. present SOCCA (Specification of Coordinated and Cooperative Activities, [103]) as language for describing the behavior of interconnected components. Subprocesses define the behavior of components, including so called traps, i.e. states which need to be reached before a different subprocess is activated for a component. The transitions between subprocesses are defined through a manager process, relating subprocesses and traps for different threads in a statechart-like manner.

2.4.6 Choreography Language Formalizations

Much work on transforming choreography languages to formal models has been reported in the literature. The motivation behind this is twofold. (i) On the one hand, a formalization defines the exact semantics of the language and therefore unambiguities in a textual definition of a language can be overcome. This reduces misunderstandings regarding the semantics. (ii) On the other hand, formalizations provide the necessary abstraction to enable reasoning on the original model or to make a verification problem decidable that was undecidable before [126]. This section lists a number of formalization approaches for the choreography languages from Section 2.3.

MSCs have been studied using communicating finite state machines by Alur et al [23, 24, 22] and by Ben-Abdallah et al. [39].

Different formalizations are available for BPMN. A special class of Petri nets, namely workflow nets [18], were used by Dijkman et al. [96]. Grosskopf has presented a mapping of an extended version of BPMN to colored Petri nets [116]. A mapping of BPMN to Communicating Sequential Processes [124] was defined by Wong and Gibbons [218]. Abstract State Machines [120] were used as formal model by Börger and Thalheim [43]. A mapping of BPMN to YAWL [16], a process definition language that is very close to reset nets [100], can be found in [70].

UML Communication Diagrams were studied using communicating finite state machines by Bultan and Fu [48, 49].

BPEL has undergone many formalization initiatives. Lohmann's mapping to Petri nets [148] handles a very large number of BPEL constructs. A comparison with other Petri-net-based approaches can be found in [153]. A mapping to π-calculus was provided by Weidlich et al. [213], to a dialect of π-calculus by Mazzari and Lucchi [158], to abstract state machines by Fahland and Reisig [107], to REO by Tasharofi et al. [202] and to SRML by Bocchi et al. [42].

Carbone et al. present a formalization for WS-CDL based on their global calculus [54].

2.5 Correctness of Choreographies

With an unambiguous definition of choreographies at hand, a number of properties of a choreography can be checked. The most basic question in interconnection models is that of *compatibility*, i.e. whether a set of interconnected roles can interact "successfully". The follow-up question then is whether for a given role r there exist roles that are compatible with r. This question boils down to *controllability* of roles. Furthermore, *conformance* answers the question whether a participant is a valid implementation for a given observable behavior description. *Realizability* answers the question whether for a given choreography there exist a set of interacting roles that show the specified behavior.

Techniques for answering these questions resort to the formal models presented in the previous section. While it is feasible to carry out formal verification manually for small models, bigger models and state spaces can only be dealt with using software tools.

The main category of software tools in this area are model checkers [32]. These model checkers typically explore the state space of a given model and check certain properties of it. For instance, it might be checked whether a valid final state can be reached from every reachable state. If a certain requirement is not met a model checker usually produces counter examples.

In addition to state space exploration techniques, Petri net theory provides additional techniques that take advantage of the net structure [92, 106]. This results in lower computational complexity for certain scenarios. Also, statements about a model in the presence of infinite state spaces can be made, which is not the case for classical model checking.

2.5.1 Compatibility

Compatibility addresses the question of whether a set of interconnected roles can interact "successfully". Unsuccessful interaction behavior could arise e.g., if different message formats are used in the collaboration and one role does not understand the message content sent by other roles. We call this *structural incompatibility*. Another source of incompatibility, which we will mainly focus on, is *behavioral incompatibility*. Imagine that a role expects a notification at some point in a process before it can proceed and none of the other roles ever sends such a notification. We call this situation a *deadlock* (see also Definition 2.7). A similar distinction under the names of *syntactic consistency* and *semantic consistency* can be found in [104].

We will again use an example from the auctioning domain for illustrating different notions of compatibility. Figure 2.15 shows an auctioning scenario with three roles. In this setting, a potential bidder must be accepted for participation before she can place her bid. Therefore, the

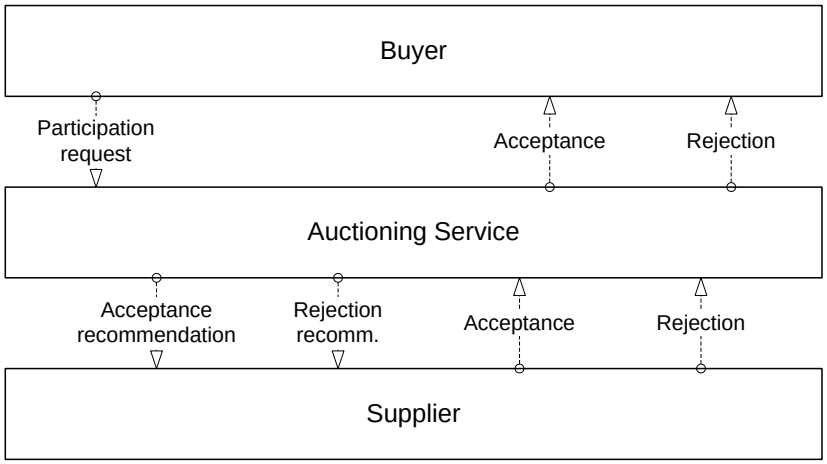


Figure 2.15: Auctioning scenario

bidder sends a participation request to the auctioning service. As a response, the auctioning service can send an acceptance or a rejection notification to the bidder. Sometimes, the seller needs to do the final decision whether a bidder should be accepted or not. The auctioning service can forward the request or already give a recommendation whether to accept the bidder or not. The seller can send a notification about his decision back to the auctioning service.

Structural Compatibility Figure 2.15 does not show any behavioral dependencies between the different message exchanges. Nevertheless, we can already judge the *structural compatibility* of this setting. In the case of structural compatibility, we analyze whether messages that can be sent or received by a role correspond to what the other roles can send or receive. The example in Figure 2.15 illustrates the case of *strong structural compatibility*: for every message that can be sent, the corresponding interaction partner is able to receive it and for every message that can be received we find a role who can send such a message.

Such a perfect structural match between roles is rather seldom. The occurrence of *weak structural compatibility* is more likely. In this case, all messages sent can be received. However, it is not required that for every message that can be received there is a role who can send such a message. E.g. the auctioning service always simply forwards the participation request although the seller could also process recommendations.

As another level of compatibility we can detect *minimal structural compatibility*. This can be applied where even weak structural soundness is too restrictive: Consider middleware platforms that are configurable in such a way that unprocessable messages are simply ignored. E.g. the auctioning service sends a notification to the seller that the bidder was informed about the decision although the seller cannot process such a message. Therefore, minimal structural compatibility merely demands that there is at least one potential message send with a corresponding message receive by another role.

Behavioral Compatibility In contrast to structural compatibility, behavioral compatibility con-

siders behavioral dependencies, i.e. control flow, between different message exchanges within one conversation. Therefore, the processes of the interacting partners are interconnected and the resulting global process is analyzed. Petri nets are used in several compatibility checking approaches. E.g. Martens bases his compatibility notion on interconnected "workflow modules" [155], a special class of Petri nets that are workflow nets with additional communication places. Workflow nets have exactly one input place and one output place and every transition is on a path from the input to the output place [18]. Workflow modules are a subclass of open nets. Figure 2.16 shows workflow modules for the roles Auctioning Service and Supplier.

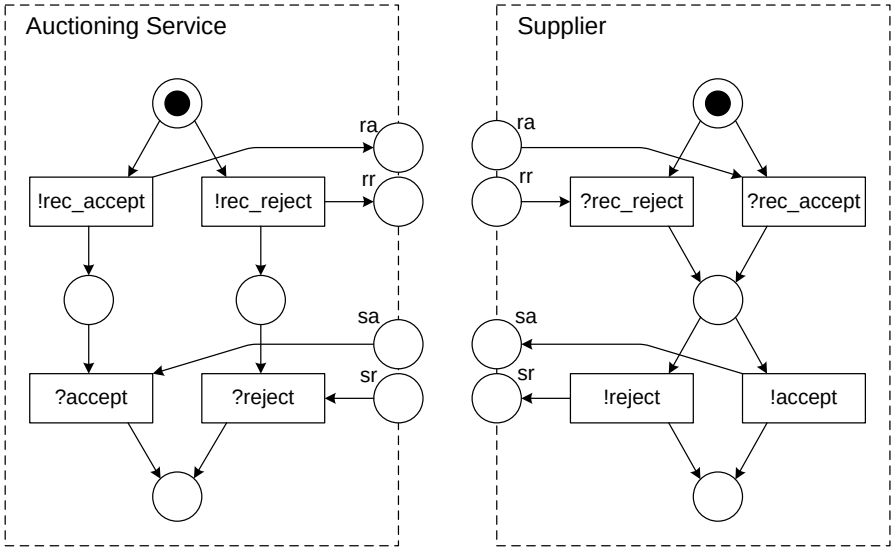


Figure 2.16: Open nets as basis for compatibility checking

Martens requires strong structural compatibility of the workflow modules. Therefore, there are corresponding places for all communication places in each module. These places are merged and a global initial place and a global final place are added. The composition is a workflow net and hence the classical soundness definition [18] can be applied. Martens uses "weak soundness", requiring that the final marking must always be reachable – while not all transitions need to be reachable. This ensures that the global process is free of deadlocks and livelocks [155].

Another correctness notion often used for checking compatibility is *weak termination* requiring that a valid final marking must be reachable from every reachable marking [152]. While the absence of deadlocks and livelocks can be checked using weak termination, it is allowed that other markings are reachable from a valid final marking.

The composition of the open nets from Figure 2.16 contains a deadlock: If the auctioning service sends an acceptance recommendation, it expects that the supplier acknowledges this by returning an acceptance message. However, the supplier has the choice to send a rejectance message instead. The auctioning service would wait infinitely in this case.

Canal et al. have also defined a compatibility notion for interacting π-processes [53]. Since interactions are atomic in the case of π-calculus, i.e. sending and receiving of messages happen at the same time, it is not possible that one process sends a message which is not consumed by

the other. The compatibility notion by Canal et al. requires that both processes complete, i.e. that no more sending or receiving action is left to be performed. A major drawback of the given compatibility notion is that it is defined for bi-lateral settings only.

Interaction soundness by Puhlmann et al. [180] is based on "lazy soundness" [181] of the global process. It is required that the process always completes, while some activities are still allowed to run even after completion. Considering these "lazy activities" is essential for coping with advanced control flow constructs such as Discrimators (cf. [17]) but leads to the fact that livelocks cannot be detected in some situations. Interaction soundness is defined for a combination of a service and its environment. We find a mixture of strong and minimal structural compatibility between the service and its environment: The environment must be able to send and receive all those kind of messages that the service can receive or send. Therefore, there must be strong structural compatibility in one direction. However, the service is not required to send and receive all those kind of messages that the environment is able to receive or send. Interaction soundness is defined on π-calculus.

2.5.2 Operating Guidelines and Controllability

Compatibility checking is an important technique when selecting interaction partners. Especially in service-oriented environments, where provided services can be discovered and bound at runtime, we have to ensure compatibility before starting to interact. Typically, the observable behavior model of the provided service is published to the broker and the service requester submits a query to the broker containing the desired observable behavior model. For enabling behavioral compatibility checking in such an environment, the service provider typically has to publish his observable behavior model to the broker. The requester then includes his own observable behavior model in the query and the broker selects a service the observable behavior model of which is compatible with the requester's one. This approach is illustrated on the left-hand side in Figure 2.17.

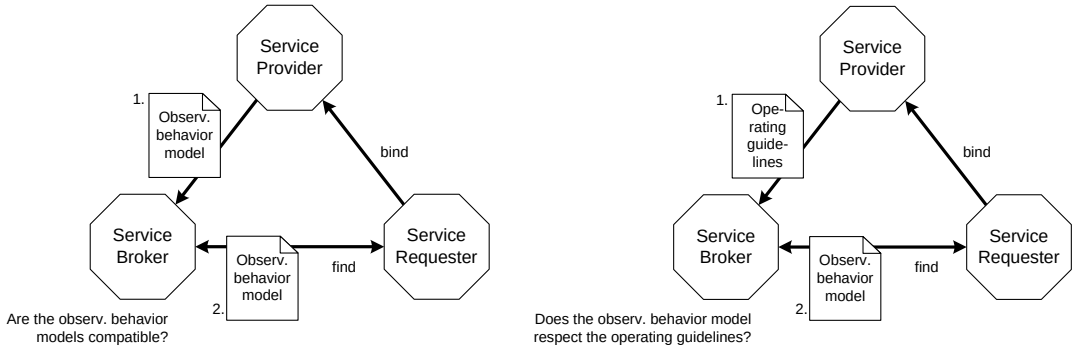


Figure 2.17: Alternative approaches for discovering interaction partners

The main problem of this approach is the computational complexity of compatibility checking. Therefore, an alternative approach was suggested by Massuthe et al. [152, 157], namely to publish *operating guidelines*. In this case reasoning does not take place on observable behavior

models. The operating guidelines of a service include all valid interaction behavior for potential interaction partners. If the consuming service respects the operating guidelines, the absence of deadlocks and livelocks is guaranteed.

Operating guidelines are annotated state machines and represent the most permissive interaction behavior for the interaction partners. The observable behavior models of the partners (also given as state machines) must then be sub state machines of the most permissive behavior.

Based on the observation that a set of roles might be compatible or incompatible the question arises whether there exist a set of roles a given role is compatible with. *Controllability* is the basis for the generation of operating guidelines. Only in case an operating guideline can be constructed for a role, this role is controllable [194].

At every moment of the construction of a controller a hypothesis about the state of a participant is made. Therefore, a hypothesis equals a set of states the participant can be in. To allow the participant to move to another state, messages can be sent or received by the controller. If it is not possible to move the participant to a valid final state, there does not exist any controller that would be compatible with the given role, i.e. the role is not controllable.

2.5.3 Conformance

The notion of conformance relates observable behavior descriptions, the specifications, with actual process implementations. The basic question addressed is whether an implementation is valid with respect to its specification. Figure 2.18 presents an example that will be used to

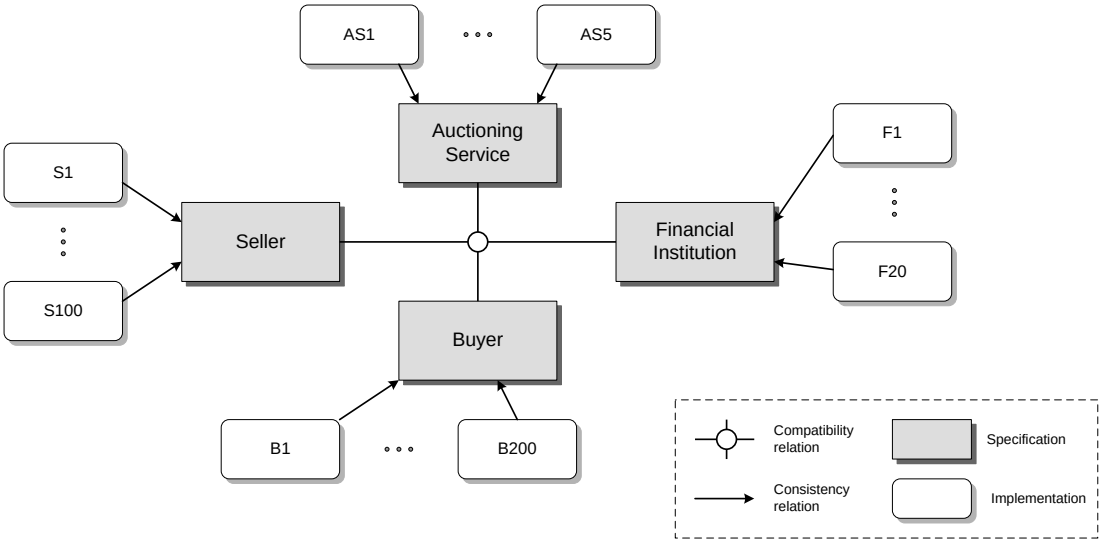


Figure 2.18: Auctioning scenario

illustrate the issue of conformance: A seller uses an auctioning service to sell goods to the highest bidder. This time, liability checks have to be carried out before a bidder is allowed to enter an auction. The liability check, in turn, is outsourced to a financial institution.

Different combinations of participants of the different roles must be able to interact successfully, i.e. they must be compatible. In order to avoid incompatibility, the participants agree on a certain desired interaction behavior. This choreography and the observable behavior models that can be derived from it are then used as contractual basis.

Figure 2.19 depicts a part of the choreography where bidders can request permission to the auction. We see that the roles bidder, auctioning service and financial institution take part in this sub-choreography. For each of these roles an observable behavior model is given in the form of an open net.

The bidder places a participation request at the auctioning service. The service then requests a liability check from the financial institution. Based on the track record of the bidder, the financial institution can recommend to accept or reject the bidder. The auctioning service is not bound to this recommendation and can freely chose whether to accept or reject the bidder.

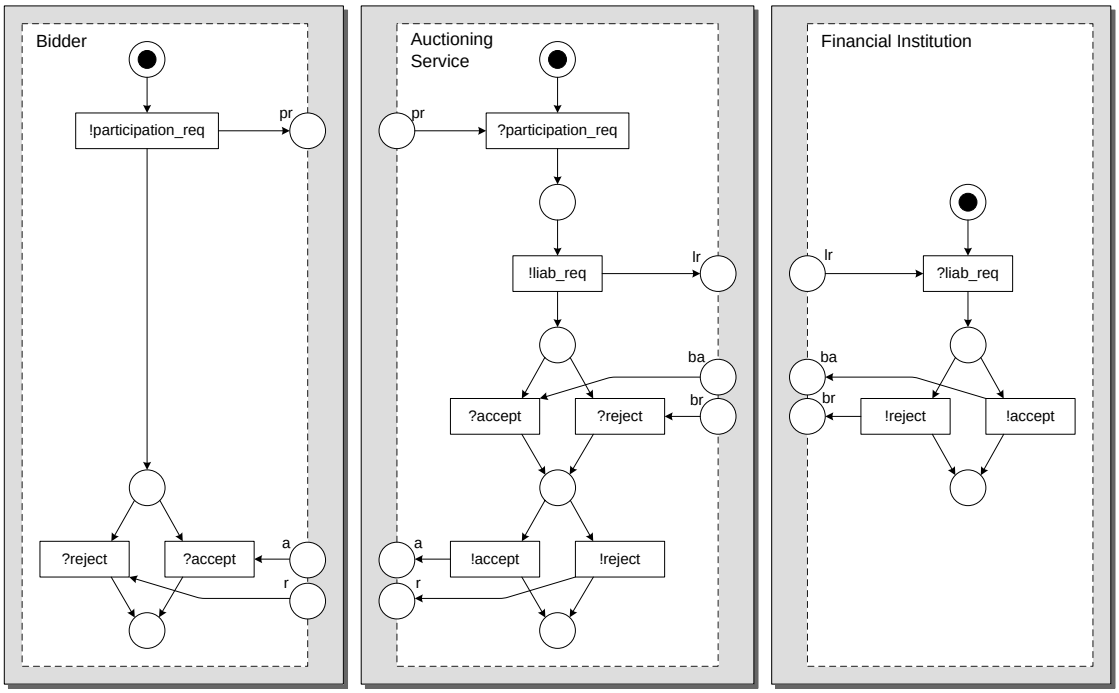


Figure 2.19: Observable behavior models: Getting a participation permission

This specification does not tell the individual participants how their internal behavior should look like. The financial institution could, for instance, lookup historical data about the supplier before coming up with a recommendation or again outsource the decision to another financial institution. Or it could have an internal decision making process possibly spanning different organizational units. No matter how the internal processes look like, we are concerned that the different participants successfully interact, i.e. that the implementations are compatible. Ensuring compatibility is a challenging and cumbersome task when dealing with a large number of participants in this auctioning scenario, involving e.g., 100 bidders, 20 sellers, 5 auctioning services and 20 financial institutions.

A remedy for this situation is the notion of conformance between observable behavior models and process implementations. The observable behavior models are the specifications for the different roles. Conformance between an observable behavior model and the actual process implementation should ensure that the given participant can interact with implementations for the other roles (provided that they in turn conform to their respective specification). That way, we can locally check whether or not a participant should be allowed to be involved in the interaction scenario. Compatibility between different implementations does not need to be checked any more.

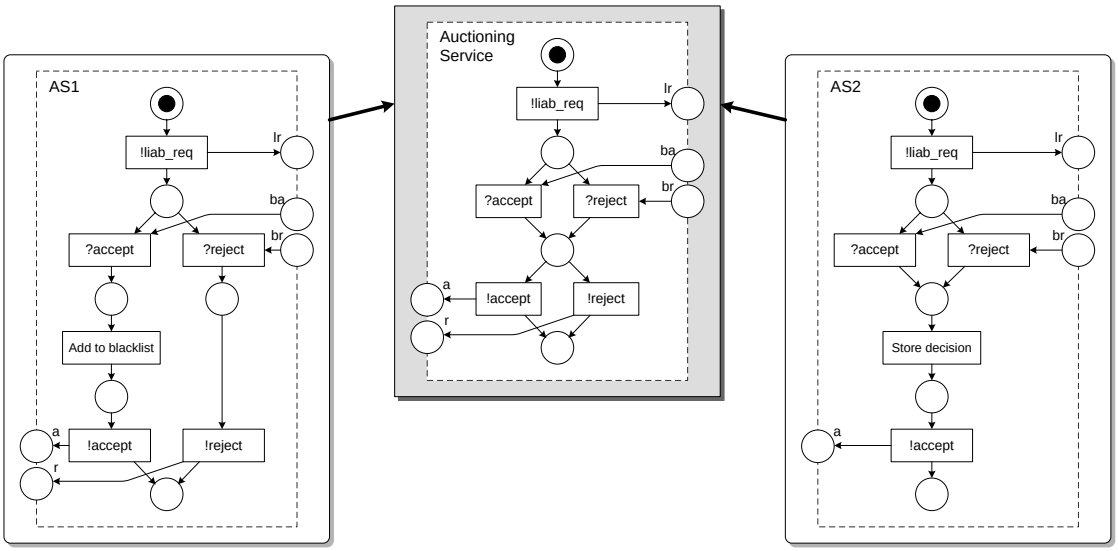


Figure 2.20: Alternative implementations for the auctioning service

Figure 2.20 presents two alternative process implementations for the auctioning service in our auctioning example. $AS1$ always follows the recommendation of the financial institution, while $AS2$ always accepts the bidder. We see that internal actions were added in both cases (e.g. "Add to blacklist" and "Store decision"). $AS1$ is structurally equivalent to the specification in Figure 2.19 but has a different control flow structure. $AS2$ is structurally different as a reject message is never sent. The question now is whether any of these implementations *conforms* to the specification.

Basten et al. introduce the notion of process inheritance in [38, 11]. The term "inheritance" refers to the object-oriented world, where a sub-class inherits attributes and behavior from a super-class and typically adds new attributes and functionality. In the case of process inheritance, it is investigated whether the process implementation inherits the behavior from the specification. Therefore, internal activities in a process implementation are not considered. At the core of this approach, bi-simulation is used to compare the two process definitions. The main idea behind bi-simulation is that a process A can simulate the behavior of process B and vice-versa. Therefore, if A is capable of doing some action then B must also be capable of doing that action and again vice-versa.

Also rooted in the object-oriented world, Ebert and Engels present an approach for relating

behavior of subclasses and superclasses [101]. The approach operates on traces. A distinction is made between *observable behavior*, requiring that all projected traces of a subclass are contained in the set of traces of the superclass, and *invocable behavior*, requiring that all traces of a superclass must be invocable on the subclass.

A set of structural rules for refining specifications into implementations were used in the Public-to-Private approach by van der Aalst and Weske [20]. While operating on the control flow structure level only, these rules ensure process inheritance. However, Basten's approach and the P2P approach do not specifically take asynchronous communication into account. In an asynchronous world, the order of sending messages becomes irrelevant as they can be buffered and overtake each other. Therefore, Lohmann et al. have extended the refinement rules [12].

In our example we see that $AS1$ is not bi-simulation related to the specification. $AS1$ cannot simulate the case that the financial institution sends an accept message after a rejectance recommendation has been received. $AS2$ is not bi-simulation related to the specification, either, as not all message receives and sends the specification is capable of can be found in these two implementations.

We see that bi-simulation is too restrictive for conformance checking. Therefore, Martens has presented an alternative relation in [156]. His technique demands that the process implementation must accept at least those messages specified in the behavioral interface and only needs to produce some of the messages specified. Using his conformance relation $AS1$ and $AS2$ would be valid implementations.

Other examples for (bi-)simulation relations are weak open (bi-)simulation for π-calculus by Sangiorgi [190] and branching bi-simulation by van Glabbeek and Weijland [205]. Busi et al. have introduced their own calculi for choreographies and orchestrations in [51, 52]. Consistency between orchestration and the specified behavior, which is given in the choreography, is shown through a bi-simulation-like relation, which is also defined by the authors. Operating guidelines can also be seen as conformance approach [152, 157]. This approach is less restrictive than bi-simulation-based approaches and even allows asynchronous communication.

As alternative name for conformance, *vertical consistency* can be found in the literature [105], as opposed to *horizontal consistency* (compatibility).

2.5.4 Realizability

Conversation models as presented in Section 2.4.2 impose the question of *realizability* of choreographies: Do a set of interacting roles exist such that they collectively realize the behavior described in the choreography? Due to the global ordering constraints of conversation models it is possible to specify behavioral dependencies that are not realizable. Figure 2.21 shows an example. Here, the choreography specifies that C is only allowed to interact with D after the interaction between A and B has happened. The intuitive reason for the unrealizability of this choreography is that C and D cannot know whether the first interaction has already happened or not.

Fu et al. have investigated realizability of conversation models in [113] and have identified three conditions for realizable choreographies.

- *Synchronous compatible condition*. The conversation model is projected to the different

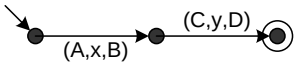


Figure 2.21: Unrealizable conversation model

roles, which are then interconnected under the assumption of synchronous communication (called the *syn-configuration*). The condition for each state in the syn-configuration is that whenever a role is ready to send a message there must be another role that is ready to receive this message.

- *Autonomous condition*. It is demanded for each role projection that there is no state where the role is ready to send *and* to receive a message. Rather, in each state the role projection is either ready to send one out of a set of messages or ready to receive one out of a set of messages. Also, it must not be possible to send or receive messages in a final state.

- *Lossless join condition*. The join of the role projections must show exactly the same behavior as the original conversation model.

As the approach uses minimal deterministic finite state machines for role projections and as it is based on traces, branching structures in the choreography are ignored. As shown in [123] and [205], ignoring branching structures leads to ignoring the moment of choice and therefore also ignoring the ownership of choice. Furthermore, realizability requires that the roles jointly do *all* behavior of the choreography. In many settings only a subset of the behavior would already be sufficient.

Realizability checking is related to conformance checking: The question of whether the joint behavior of interacting roles realizes the choreography equals the question of conformance between an implementation and a specification. The additional challenge of realizability originates from the fact that the potential role behaviors must be identified from an infinite set of behaviors.

Realizability will be studied in greater detail in Section 5.4, where different notions of realizability will be distinguished and the issue of branching structures will be investigated.

Chapter 3

Evaluation of Choreography Languages

The previous chapter has presented prominent choreography languages, choreography formalisms and correctness notions. This chapter relates this related work to the requirements and research questions from Section 1.3.

The three choreography language requirements *expressiveness*, *ease of use* and *ease of adoption* were described in a very abstract manner. Therefore, we are going to introduce more concrete requirements in this chapter to enable an assessment. These requirements are derived from the motivating example (cf. Section 1.1), choreography design approaches and existing comparison frameworks.

Business processes are typically considered on two different levels. On the one hand, *conceptual* process models are used for identifying process improvement potential, for mere documentation purposes or for doing analysis such as cost calculations. On the other hand, *technical* (or implementation-centric) process models can directly be interpreted by process execution engines. These two levels must also be distinguished in the choreography space. Choreography models on a conceptual level mainly support and reflect choreography design decisions. While enumerating roles, interactions and behavioral dependencies between them, the models are not necessarily used for information system implementation later on. Technical concerns are not the main focus of these models. Meaningful English names for interactions are often sufficient. Sometimes, interactions are not even broken down to elementary interactions and stay on the level of complex interactions. In contrast to this, choreography models on a technical level mainly target elementary interactions and their dependencies without considering higher levels of abstraction. Links to the implementation level of information systems, especially in the form of software services, are essential.

These two levels must also be distinguished when comparing the suitability of choreography languages. Due to their different target user groups and concerns, requirements are slightly different between the two types. Therefore, we will distinguish three kinds of requirements for choreography languages: those that apply to conceptual choreography modeling only (Section 3.1), those that are essential for both conceptual and technical choreography modeling (Section 3.2) and those that only apply to technical choreography modeling (Section 3.3). While the

first category is mostly motivated through choreography design approaches, the requirements of the second category are derived from real-world choreography examples as presented in Section 1.1. The third category deals with technical details that need to be agreed upon before configuring information systems. The requirements were identified in and validated through industry projects carried out at SAP Research.

Section 3.4 carries out the assessment using the three groups of requirements. It identifies the white spots of these languages that will serve as starting point for the remainder of this thesis.

The problem statement from Section 1.3 also put up a research question regarding the formal verification of choreographies. Section 3.5 will investigate in how far existing correctness notions and choreography formalisms can be used to answer the question "Which choreographies can be implemented by a set of participants?" As this question is rather abstract, we are going to further refine it in this chapter.

3.1 Requirements for Conceptual Choreography Modeling

Choreography modeling on the conceptual level should allow designers to focus on the essence of choreography models, which are thereby free of implementation- (or technology-) specific details. In this way, a choreography language enables the functionality of choreography models to be understood and validated against their intended business requirements, without that task being compromised by implementation details or choices. The risk of the language not being based on the conceptual level, and driven by implementation concerns, is that it loses its currency at the design stage where it is crucial that the scenarios are carefully captured and communicated by business analysts rather than technicians.

Since a model at the conceptual level is intended for communication with business analysts, a choreography language must be endowed with graphical constructs. Usability of the language must be given through a simple and elegant set of first-class constructs, and on the other hand not awkwardly visualizing specification details that would be more suitable in text form. For example, branching conditions are suitably expressed through textual constraints, while enablement or disablement dependencies between interactions should be graphically expressed.

C1. Graphical notation. A choreography language on the conceptual level must have a graphical notation for representing the notions of roles, interactions and enablement/disablement dependencies, while allowing to define branching and repetition conditions in textual form.

Many choreographies involve many roles and a large number of interactions. Here, we run into the problem that choreographies become very complex and hard to manage. Especially reaching agreement among all participants makes choreography design a challenging task. Many choreography approaches acknowledge this complexity and propose means to cope with it. We can distinguish between (i) viewpoints and abstraction concepts and (ii) design methods. The first category follows a separation of concerns [201] and divide and conquer [59] strategy to break choreographies into manageable pieces and provide a framework to relate the different pieces. The second category deals with the process of creating choreographies, i.e. the order of

modeling decisions and modeling steps.

Viewpoints and Abstraction Concepts

Achieving separation of concerns by using different viewpoints is a common strategy in the areas of information systems and BPM in particular. For instance, ARIS includes the following viewpoints: functional (enumerating activities), organizational, data, service and process, while the latter serves as glue between the former [191]. Different viewpoints are also advocated in the Fundamental Modeling Concepts framework (FMC [141]) and in the Reference Model of Open Distributed Processing (RM-ODP [132]).

Especially the structural view, or role-based view, is a viewpoint that is used in many choreography approaches. The different roles are identified as well as their interconnection. Role-based models help to find the right scope of a choreography and serve as initial big picture.

C2. Structural view. Choreography languages on the conceptual level must support the representation of a structural view, where only roles and how they are interlinked are shown. Therefore, a choreography language must allow models that do not contain behavioral dependencies.

Typically, additional abstraction layers are introduced to manage the complexity. With different levels of granularity at hand, the involved participants can agree on individual artifacts and subsequently refine them in a step-by-step manner.

In this context, high-level behavioral models are often used. As the collaboration between different roles specified in a choreography is related to reaching a certain goal, several steps can be identified leading to the goal. E.g. the final goal of a bidding scenario is that the offered goods are sold, paid for and delivered to the bidder with the highest bid. As intermediate steps we can distinguish between the initial setup of the auction, the actual bidding phase, the delivery phase and the payment phase. These sub-choreographies collectively realize the overall choreography.

Since steps might not be clearly separated and therefore overlap, we might alternatively concentrate on the outcome of these steps. Outcomes such as “auction is set up” or “bidding phase is over” are sub-goals or milestones, showing the path to the final goal. Similarly, we can define the dependencies between milestones: The auction has to be set up before the bidding process can be finished.

At the lowest level of abstraction, the detailed choreography model can be found. Here, the elementary interactions that are needed to proceed from one milestone to another or to realize a certain sub-choreography are related. That way, the overall choreography is partitioned into different scenario models. One or several scenario models show the interactions and their dependencies that need to happen between two milestones.

C3. Modularity of choreographies. In order to manage large choreographies, a choreography language on the conceptual level must support to partition a model into manageable pieces.

Design Methods

The motivation behind many choreography languages is to enable a model-driven approach for service design and implementation. These top-down approaches, like presented in [95] and advocated in [136], propose choreographies as a starting point for generating observable behavior models for each service which are then the skeletons for implementing new services or for adapting existing services [135]. This top-down approach is also applied to the creation of choreography models. For instance, Colombo et al [58] propose a methodology for service composition that starts with the definition of so-called *social models* that capture business entities and their dependencies. In the second phase of the methodology, a process model capturing the behavior of a service composition is constructed. This model is derived from a set of ECA rules and it is encoded as a finite state machine.

A number of approaches see the investigation of commercial transactions and the business semantics behind them as starting point for choreography modeling. The Design and Engineering Methodology for Organizations (DEMO [94, 93]) follows this approach and proposes the hierarchical decomposition of commercial transactions as preparation for the following process modeling step. The speech act theory [29, 196] relates information exchanges and intentions of the roles. By considering the semantics of the information exchanges, only certain refinements into more detailed information exchanges and finally message exchanges can be allowed. This strategy is applied in [122], where the Semantic Object Model (SOM [110]) is used in the context of choreography modeling. Through the refinement steps and the semantics of the message exchanges, a certain order of message exchanges can be identified. Detailed modeling of the behavioral constraints must then be done on a lower level.

It turns out that some application scenarios cannot be covered using top-down approaches [87]. Existing services and processes might be the starting point for identifying already existing choreographies or for creating to-be choreographies. The following three use cases require a bottom-up method for choreography design:

1. In the case of *choreography identification*, different participants have working process implementations and already use them to collaborate with each other. However, every participant only knows about the interactions he is directly involved in. Therefore, the goal is to identify the overall interaction behavior, i.e. the choreography, the participants already engage in so that everybody has a global view of the collaboration. This use case requires that the actual choreography is extracted from the existing collaborating processes. This can be done either based on the process implementations or based on the observable runtime behavior of the systems involved.

2. In the case of *choreography context expansion*, choreographies that are limited to a certain business context need to be extended. This can broaden the reach of the choreography, i.e. make the choreography applicable to a broader context.

3. In the case of *collaboration unification*, different observable choreographies exist for the same domain. A typical reason for the evolution of such "islands of collaboration" is that there are disjoint groups of collaborators each of which has its own history for the

interaction protocol. Now the goal is to enable the collaboration between participants from different islands through a unified choreography for all participants.

Dorn et al. present a comprehensive survey on approaches in the area of B2B integration [98]. The approaches are classified using the Open-EDI reference model [3] and distinguish business models, process models and artifacts that are ready for deployment.

All top-down choreography design approaches include the step-wise refinement of complex interactions into finer-grained interactions with dependencies between them until finally a message level is reached. While such refinement takes place in top-down approaches, bottom-up approaches aggregate individual interactions into more complex interactions.

C4. Decomposition of interactions. Choreography languages on the conceptual level must support to reflect hierarchies of interactions.

During choreography design it might turn out that some parts of a choreography have been agreed upon before in a different setting and can be reused unchanged. While an integration of behavioral dependencies of different models is typically well understood, an integration on the role level is often rudimentary. The payment phase in the auctioning example can illustrate this case: Payment between the roles payer and payee is a well understood complex interaction that can be reused in many different contexts. In an auctioning scenario, the payer role is played by the bidder and the payee role by the seller. In another context, the seller becomes the payer and the auctioning service the payee, e.g. if the seller needs to pay for using the auctioning service.

C5. Reusability of choreographies. Choreography languages on the conceptual level must support the reuse of subchoreographies in different choreographies, including means for integrating the behavioral dependencies as well as the roles.

3.2 Requirements Derived from Choreography Examples

Key aspects of choreography languages, as for all modeling languages, are *expressiveness* and *suitability* [139]. Expressiveness refers to the mere *ability to express* certain situations using the language, while suitability refers to *how easy* it is to express typical real-world scenarios.

With the rise of choreography languages in the early 2000s, a growing need for comparability of them was identified. This resulted in several initiatives, each producing a set of recurring scenarios, or patterns, that can be used to assess languages and systems. The idea of pattern-based assessment of languages was already used in the Workflow Patterns initiative [17]. However, the Workflow Patterns only concentrate on control flow structures within process models and do not capture scenarios where two or more participants engage in complex interactions.

This gap gave rise to the Enterprise Integration Patterns by Hohpe and Woolf [127] and the Service Interaction Patterns by Barros et al. [36]. The Enterprise Integration Patterns operate on a rather technical level. They enumerate typical practical solutions for system integration styles, messaging systems and channels, message construction, routing and transformation and messaging endpoints. The pattern catalog by Aldred et al. [21] has a similarly technical focus.

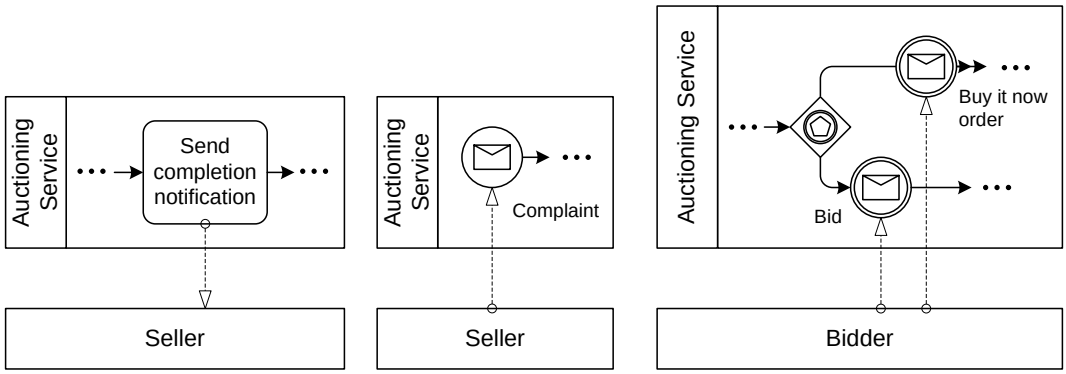
The Service Interaction Patterns are a catalog of choreography patterns. There is some overlap between these patterns and the Workflow Patterns. Often, a Service Interaction Pattern requires a certain Workflow Pattern, while adding aspects that are specific to choreographies. The catalog of patterns consists of four categories. The distinction is based on the number of participants involved (bi-lateral vs. multi-lateral interactions), the number of messages exchanged (single-transmission vs. multi-transmission interactions) and if in the case of two-way interactions the receiver of a response is necessarily the same as the sender of the request (round-trip vs. routed interactions). The resulting categories therefore are: (1) Single-transmission bilateral interaction patterns, (2) single-transmission multi-lateral interaction patterns, (3) multi-transmission interaction patterns and (4) routing patterns.

All patterns can be found in the auctioning domain. Therefore, we are going to use corresponding examples to illustrate the patterns. A BPMN representation will be given for these examples. In some illustrations three dots are used to represent an unknown number of pools. The three dots are not part of BPMN and the deficiencies of BPMN regarding unknown number of participants in a conversation will be addressed later in more detail (cf. 2.3.2).

Single-transmission bilateral interaction patterns

Pattern 1: Send In the case of *Send* a one-way message exchange between two participants is considered from the perspective of the sender. There are different flavors of this patterns, considering e.g. the moment when the sender selects the receiver: Either the receiver is already known at design-time of the choreography or only during the execution of a conversation.

Figure 3.1(a) illustrates an example where the auctioning service sends an auction completion notification to the seller as soon as the auction has successfully finished.



(a) Send (b) Receive (c) Racing incoming messages

Figure 3.1: Patterns 1, 2 and 4

Pattern 2: Receive This pattern also describes a one-way interaction between two participants, but this time seen from the perspective of the receiver. In terms of message buffering behavior of the receiver, we can distinguish two cases. Either messages that are not waited for are discarded or they are stored until consumption (or timeout).

In the example in Figure 3.1(b) the auctioning service receives a complaint from the seller. E.g. the seller's offer has been removed and the seller now wants to know why.

Pattern 3: Send/receive A participant sends a request to another participant who then sends a response back. Both messages belong to the same conversation. Since there could be several Send/receive message exchanges happening in parallel, corresponding requests and responses need to be correlated.

Imagine a seller requesting auction creations for different items. The different request/response pairs belong to different conversations. In this situation the seller must be able to tell which acknowledgement belongs to which request. Therefore, correlation information must be placed inside the messages. E.g. the request could carry a request ID which is then also contained inside the response.

Single-transmission multi-lateral interaction patterns

Pattern 4: Racing incoming messages A participant is waiting for a message to arrive. Different other participants have the chance to send a message. The first message arriving will be processed. The type of the message sent or the category the sending participant belongs to determines how the receiver processes the message. The remaining messages may be discarded or kept for later consumption.

Figure 3.1(c) shows a scenario where an item is offered in an auction and as fixed price offer at the same time. Users interested in that item can choose whether to buy it straight away or participate in the auction. However, once the first bid has been placed, the fixed price offer is not available any longer.

Pattern 5: One-to-many send A participant sends out several messages to different other participants in parallel. It might be the case that the list of recipients is already known at design-time of the choreography or alternatively selection takes place in the course of the conversation.

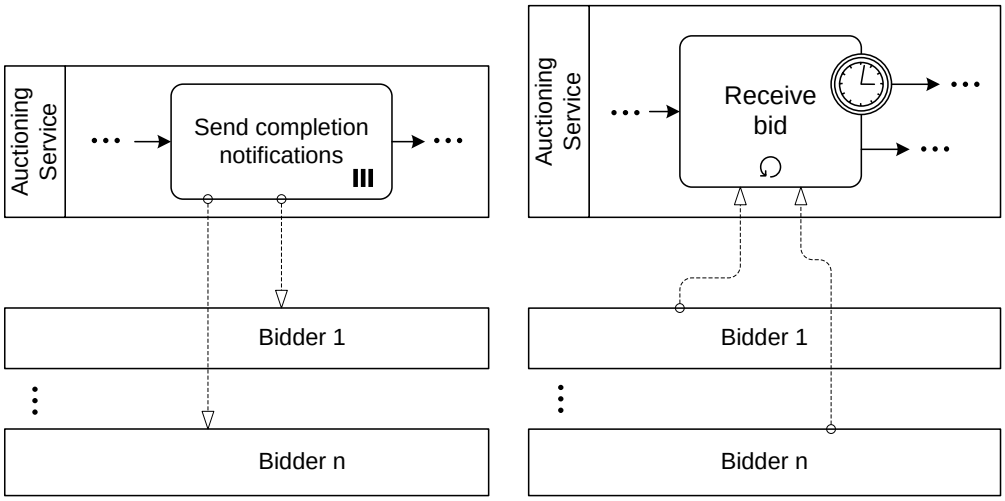
An example for this pattern is shown in Figure 3.2(a): Once an auction is over, all those bidders who have not placed the highest bid are notified that they did not purchase the item. References to similar items might be included in the messages so that the bidders can try again in a different auction.

Pattern 6: One-from-many receive A participant waits for messages to arrive from several other participants. Typically, the receiver does not know the number of messages that will arrive and stops waiting as soon as a certain number of messages has arrived or a timeout occurs.

Figure 3.2(b) illustrates a core part of auctions: Different bidders can place bids for a particular item until the auction is over.

Pattern 7: One-to-many send/receive A participant sends out requests to other participants and waits for responses. Typically, not all responses need to be waited for. The requester rather waits for a certain time or stops waiting as soon as enough responses have arrived.

After an auction is over and the buyer has been identified from the set of bidders, the seller selects a shipper for delivering the item to the buyer. Depending on where the buyer is located



(a) One-to-many send (b) One-from-many receive

Figure 3.2: Patterns 5 and 6

(e.g. rural area vs. metropolitan area, same country vs. another European country vs. outside of Europe), the shipper charges different prices. The seller therefore sends price requests to all shippers available and waits for price quotes to be returned.

Multi-transmission interaction patterns

Pattern 8: Multi-responses A participant sends a request to another participant who sends multiple messages back. An important question in this scenario is how the requester knows that there are no more messages to be expected. One option is that the messages contain information about whether there will be more messages or not. Another option is that the last message is of a special type. Finally, also a timeout could be used to stop waiting for further messages.

Once the shipment of the purchased item has been initiated, a buyer can register for notifications for the current delivery. The shipper sends status updates. Finally, a delivery notification is sent once the item has reached its final destination.

Pattern 9: Contingent requests A participant sends a request to another participant. If this participant does not answer within a given time, the request is sent to a second participant. Again, if no response comes back, a third participant is contacted and so on. Delayed responses, i.e. responses arriving after the timeout has already occurred, might be discarded or not.

Imagine an example where an auction has finished and the buyer of the item turns out not to be able to purchase the item. E.g. the buyer has become insolvent. As special offer for the seller, the auctioning service finds an alternative buyer for the item. As the other bidders have already shown interest in that item by placing bids, they are subsequently asked whether they want to purchase the item for the auction's final price. If the bidder with the second-highest bid rejects or does not answer in time, the bidder with the third-highest bid is asked and so on.

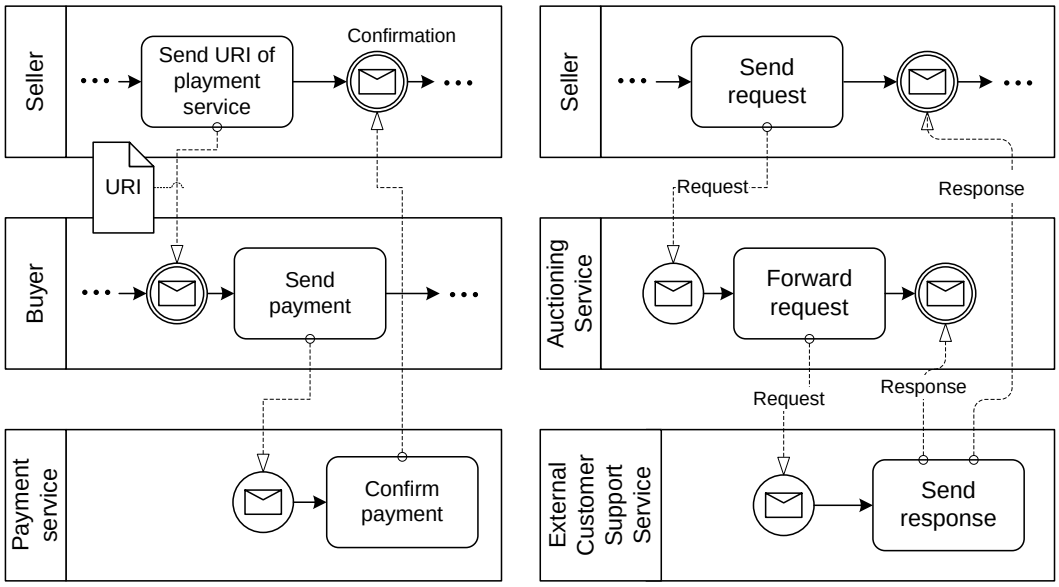
Pattern 10: Atomic multicast notification A participants sends out notifications to several other participants who have to accept the notification. Sometimes only one participant is required to accept it, sometimes some of the participants and sometimes all of the participants are required to accept it.

Imagine a seller frequently offers a combination of items in a single auction. The seller acts as single point of contact to the auctioning service and the buyer. However, before the auction is actually created the seller notifies all subcontractors. As soon as a certain number of subcontractors have accepted this notification, the seller actually starts interacting with the auctioning service.

Routing patterns

Pattern 11: Request with referral A participant *A* sends a message to another participant *B* containing a reference to participant *C*. Although *B* does not need to know *C* in advance, *B* can now interact with *C*. This pattern describes the concept of *link passing mobility*.

Figure 3.3(a) shows the integration of an external payment service. The seller selects a payment service and sends the service's reference to the buyer. The buyer, who previously might not have known the service, directly interacts with it. Finally, the payment service sends the confirmation that the payment was successfully completed to the seller.



(a) Request with referral

(b) Relayed request

Figure 3.3: Patterns 11 and 12

Pattern 12: Relayed request A participant *A* sends a request to another participant *B* who forwards it to a third participant *C* who will actually interact with *A*. However, *B* always gets

copies of the messages exchanged in order to be able to observe the conversation.

Figure 3.3(b) depicts an example where the auctioning service outsources customer support activities to an external customer support service. As soon as a request from the seller comes in, the auctioning service forwards the request to the external service. For all responses sent by the external service to the seller, the auctioning service receives a copy of that response. That way, the auctioning service can monitor the quality of the responses as well as the response times for different requests.

Pattern 13: Dynamic routing A routing order is attached to a message. This order determines where the message needs to be sent to next by the recipient of the message. However, the recipient might change this routing order by adding further recipients or changing the order in which participants get the message.

As an example we consider how complex customer complaints are processed. If the customer support agent first assigned to a certain complaint is not sure what to do with the complaint, she might forward it to another agent. She might as well attach a number of agents to the complaint if it touches various aspects and an expert for every aspect needs to be involved.

The original description of this pattern in [36] is quite imprecise: "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'." E.g. the term "dynamic condition" remains unclear. This pattern is unsuited for assessing languages and will not be considered in the remainder of this thesis.

As many of the patterns have considerable overlap, we are not going to use the full set of patterns in our requirements framework. A number of key concepts can be extracted that must be supported by choreography languages both on a conceptual and a technical level.

R1. Multiple roles. More than two roles can be involved in a choreography. The higher the number of roles, the more important the choreography becomes and the more challenging it becomes to capture the complex interaction dependencies. A choreography language must support an unlimited number of roles in one choreography. This requirement relates to the Service Interaction Patterns 4–7 and 9–12.

R2. Ownership of choices. Alternative branches are a common scenario in choreographies. In our auctioning example, the bidder only enters into the payment and delivery phase in case she has placed the highest bid. Otherwise, the conversation ends for the bidder. Decision points in a conversation can be decided by one participant only or a number of participants decide based on previous information exchanged in the conversation. An example for the first case is that the auctioning service decides which bidder wins. An example for the second case would be the decision about whether another bid can be placed or not: All participants involved know when the auction stops and can act accordingly – provided that they have a similar understanding of time. We can conclude that a choreography language must supporting branching and the notion

of decision ownership. This relates to the Service Interaction Pattern 4 and the Workflow Patterns Explicit Choice and Deferred Choice [17].

R3. Multiple participants per role. Multiple bidders can be involved in one auction. Here, the number of bidders in one auction is even unlimited and therefore not known at the design time of the choreography. While the role concept itself allows different participants filling the same role in different conversations, e.g. different sellers in different auctions, this is not sufficient to fulfill this requirement. It demands that a choreography language must support a (potentially unknown) number of participants of the same role in *one* conversation. This requirement relates to the Service Interaction Patterns 5–7.

R4. Participant reference passing. The selection of participants is often done at design-time of choreographies, e.g. an auctioning choreography might be bound to one particular auctioning service implementation. In other cases, selection is done at runtime. In both cases it must be ensured that other participants are made aware of the selection if they are to interact with these participants as well. In our example the auctioning service has to pass on the reference of the successful bidder to the seller. Otherwise, the seller would not know where to send the payment details to. A choreography language must support the notion of participant reference passing. This requirement refers to the Service Interaction Pattern 11.

R5. Cancellation. Choreographies not only include the "happy flow" of interactions but also contain the desired behavior in case a participant changes his mind and wants to cancel certain parts of a choreography. This builds on the Cancel Activity pattern [17] and the Reroute pattern [169]. In simple cases, a cancellation can be stated as alternative to a confirmation or acceptance message (cf. Service Interaction Pattern 4). In more advanced cases, cancellation must be supported while a conversation has already continued. An example would be that the auctioning service cancels an auction during the bidding phase.

R6. Time constraints. As most important non-functional aspect, time constraints must be properly reflected in choreographies. For instance, offers might only be valid for a certain time period or a response is expected within a certain timeframe. It must be possible to express desired reactions to timeouts.

Another set of patterns was collected by van der Aalst et al. [13], concentrating on different types of sending and receiving, also in the presence of concurrency, decision making and correlation configurations.

3.3 Requirements for Technical Choreography Modeling

Choreography modeling on a technical level comes with an additional set of requirements. Even if all roles have already been identified, interactions have been refined down to the message level and behavioral dependencies have been set between them, a choreography is not ready to be used for actual electronic integration. A number of technical configurations have to be made

to guarantee interoperability of the information systems involved. These most importantly include the specification of message formats, correlation and exception handling.

T1. Message formats. Choreography initiatives such as RosettaNet show that it is possible and fundamental to agree on the format of messages exchanged. Message formats not only specify the desired content of messages but also the concrete syntax or data formats that are to be used. A choreography language must allow to define the message formats expected for an interaction.

T2. Correlation. It cannot be expected that each participant is involved in at most one conversation at a time. It must be possible to distinguish different conversations and to relate an incoming message to the messages previously sent or received. Such correlation is typically realized by placing correlation information into messages. In the auctioning example, correlation information could be an auction id or a bid id. In general, the service providers must agree on where in the message such information can be found and who is responsible to generate which identifier. By checking the equality of an id included in a message and an expected id, correlation is carried out. A choreography language must support the definition of such correlation configurations.

T3. Exception handling. There might be purely technical reasons why messages are not sent on time or contain the wrong content. For these cases, control flow handling faults and exceptions must be defined. Reactions in the form of cancellation messages or a simple termination of a conversation might be necessary. Therefore, a choreography language must allow the definition of exception handling.

Setting up a fully specified technical choreography is a challenging task. In order to take advantage of the work previously done in changing technical environments, it must be possible to reuse technical decisions while replacing others. This mostly applies to message formats. When tightly integrating message format specifications into choreographies, the reusability of the choreography is limited.

Currently, the Web Service Description Language (WSDL, [56]) is a widely used standard to describe the structural interface ("port type" as a collection of "operations") of a web service. A WSDL definition includes the definition of message formats, usually using XML Schema, as well as the information needed to interact with a physically deployed service—on the wire manifestation of messages and the transport protocols ("binding") as well as the physical endpoints ("ports") [62]. Standards such as RosettaNet show that it is possible to standardize the exchanged messages, but that it is not possible to standardize the interfaces to be used. Therefore, WSDL's ability to standardize messages definitions is important in choreography design, whereas the description of operations in an interface as well as port and binding information are not needed, because their use would force the choreography adopters to follow these technical realizations.

T4. Interchangeability of technical configurations. Changes in port type or operation names should not require changing the choreography. Also other technical details, e.g. whether a service is realized using one port type or two port types or whether two messages are exchanged

in a synchronous request/response cycle or asynchronously, should not require major changes in the choreography. Sometimes it is not possible to agree on one message format for a particular interaction. In these cases, message mediation is necessary. There should at least be an extension point to plug in corresponding configurations. A choreography language must support interchangeability of technical configurations.

BPEL is the de-facto standard to implement business processes based on web services. Round-tripping between specifications given in the form of choreographies and implementations in the form of BPEL processes must be facilitated through an integration between the choreography language with BPEL.

T5. Integration with web service orchestration languages. Choreography languages must allow an integration with BPEL, including easy generation of BPEL processes out of choreographies and extracting choreographies out of existing interacting BPEL processes.

3.4 Assessment of Choreography Languages

Each language presented in Section 2.3 can be evaluated along the requirements catalog from the previous sections. This section presents this evaluation. It will be distinguished whether there is full support for a requirement, partial support or no support. This rating scheme is taken from the Workflow Patterns initiative[1] and analogously applied in this section. Full support means that there is an explicit modeling construct realizing a requirement or that a combination of only a few constructs are necessary to realize it. No support means that either the language does not include the concept demanded in the requirement at all or that difficult workarounds would be necessary to realize the requirement. Partial support indicates that only certain parts of a requirement are fully supported while others are not.

Message Sequence Charts (MSC)

Due to its standardized graphical notation, MSCs support requirement *C1*. However, a structural view is not supported (*C2*), nor are modularity of choreographies (*C3*), decomposition of messages (*C4*) and reusability of choreographies (*C5*).

MSCs have very limited expressiveness for many choreography concepts. While roles and interactions are present, only simple sequences of communication activities are defined in an MSC. MSCs provide direct support for expressing the Service Interaction Patterns *Send*, *Receive* and *Send/receive*. All other patterns are not supported by MSCs as they lack constructs for expressing branching, repetitions as well as multiple participants of the same type (where the number of participants is only known at runtime). Therefore, only requirement *R1* is supported. All other requirements are not supported.

[1] See `http://workflowpatterns.com/`

Business Process Modeling Notation (BPMN)

Having a strong focus on an intuitive graphical notation, BPMN supports requirement *C1*. By collapsing pools and only showing message flow between them, structural views can be modeled in BPMN (*C2*). BPMN offers two mechanisms for relating different models: isolated subprocesses that are specified in a separate diagram and link events connecting multiple diagrams. Therefore, modularity can be achieved (*C3*). However, while there is reusability on the activity level, there is no reusability on the role level (*C5*). While BPMN includes decomposition facilities for activities, this is not the case for message flow (*C4*). Only one level of abstraction of messages can be specified in a BPMN model.

Involvement of two or more participants can be represented by a corresponding number of pools (*R1*). Branching structures can be reflected in BPMN through decision gateways. The distinction between data-based XOR gateways and event-based XOR gateways allows to properly reflect ownership of choices (*R2*). In contrast to this, BPMN does not allow the distinction between at most one and potentially many participants per role (*R3*). BPMN's capabilities for data flow are also not sufficient to reflect participant reference passing (*R4*). The notion of cancellation is natively supported through intermediate events that can be attached to subprocesses (*R5*). Timing is also supported in BPMN, namely through timer events (*R6*).

As BPMN is a modeling language that rather operates on the conceptual level, many of the technical requirements are not met. However, there are attributes for defining concrete message formats (*T1*). Correlation configurations, in turn, cannot be set (*T2*). Again, exception handling is supported via error events (*T3*). Interchangeability of technical configurations is not given (*T4*). An integration with existing service orchestration languages is given through the extensive work on mapping BPMN to BPEL [168, 167]. While mapping in both directions is possible for a large number of constructs, different coverage of concepts and semantic differences between corresponding constructs do not allow for complete round-tripping [212]. Therefore, we conclude that there is only partial support for requirement *T5*.

Business Process Schema Specification (BPSS)

BPSS does not a have graphical notation (*C1*). As the roles and their interactions could be listed without giving any behavioral dependencies, BPSS supports requirement *C2*. Modularity and reusability of choreographies are not given (*C3* and *C5*). Decomposition of interactions is not possible, either (*C4*).

In contrast to earlier versions of BPSS, the current version 2.0.4 supports multi-lateral choreographies (*R1*). Although choices can be specified in a BPSS choreography, it remains unclear who is responsible for evaluating the conditions (*R2*). Multiple participants per role are not supported, either (*R3*), nor is participant reference passing (*R4*). Cancellation is not supported (*R5*). However, timing issues can be specified in detail in BPSS (*R6*).

While BPSS is technology-independent, WSDL files can be referenced in the Collaboration Profile Agreement (*T1*). Exception handling can also be specified (*T3*). Correlation configurations and interchangeability are not given (*T2*, *T4*).

There has been a mapping of individual business transactions to executable BPEL code [125]. However, as business collaborations are not covered, we conclude that there is no support

for an integration with orchestration languages (*T5*).

UML Communication Diagrams

Although UML communication diagrams come with a graphical notation, behavioral dependencies are represented textually. Therefore, requirement *C1* which also demands that enablement and disablement of interactions can be modeled graphically is only partially supported. When omitting the sequence expressions, communication diagrams act as structural views (*C2*). Sequence expressions are of a block-structured nature. However, decomposition is not supported as complex interactions cannot be modeled (*C4*). Modularity and reusability are not given (*C3* and *C5*).

An arbitrary number of roles can be present in one communication diagram (*R1*). Alternative branches are realized using guard conditions and the evaluator of the condition is not explicitly specified (*R2*). Participants and roles are distinguished in communication diagrams and multiple participants of the same role can take part in one conversation. However, as the number of participants must be known at design-time, requirement *R3* is only partially supported. Participant reference passing is not supported at all (*R4*). Only simple cases of cancellation can be represented using communication diagrams, canceling a whole choreography regions through a cancellation message cannot be modeled (*R5*). Time constraints cannot be modeled at all (*R6*).

Communication diagrams inherit many of the attributes available in the UML meta-model. However, binding concrete message formats, for instance specified in WSDL, to interactions is not possible (*T1*). Correlation configurations and exception handling cannot be defined, either (*T2* and *T3*). Technical configurations cannot be set (*T4*) and an integration with common orchestration languages is not given (*T5*).

Business Process Execution Language (BPEL)

BPEL does not come with a standardized graphical notation (*C1*). BPMN is often advocated as graphical frontend for BPEL, but an integration is challenging, as Section 4.3 will detail. A structural view is not available in BPEL (*C2*). In many cases, it is also impossible to derive such a view, since discovery information does not form a part of BPEL. BPEL refers to port types, not to ports. Therefore, a given set of BPEL processes with matching port types does not guarantee that these BPEL processes are actually interconnected.

Modularity is partially given due to the recursive nature of BPEL processes: BPEL processes implement web services that can in turn be invoked by other BPEL processes. However, in this scenario one role would be implemented by different processes. This is counterintuitive in the choreography context where roles are a very central concept. Reusability and modularity can also be achieved through WS-BPEL 2.0 Extensions for Sub-Processes (BPEL-SPE [140]). All in all, we conclude that BPEL only partially supports *C3* and *C5*. BPEL does not include the notion of complex interactions and therefore decomposition is not supported *C4*.

The role concept is realized through partner links in BPEL. Ownership of choices is realized through the distinction of `if` and `pick` in BPEL (*R2*), in analogy to data-based and event-based gateways in BPMN. Multiple participants per role, i.e. multiple services per partner link, are possible in BPEL since version 2.0 [108] (*R3*). Here, partner links can be defined within scopes.

In combination with the parallel `forEach` construct, a partner link can be used with multiple services concurrently in the same process instance. Endpoint reference sets can be handled through corresponding XML schema types. Therefore, realizing the "One-to-many send/receive" pattern [36] is possible. However, relying on external type systems, sets or lists are not first-class citizens in BPEL and therefore working with endpoint reference sets is cumbersome.

Service endpoints references can be stored in BPEL variables. This storage allows to receive endpoint references from other services or sending endpoint references to other services. Since an `assign` activity can be used to assign an endpoint reference to a partner link, it is possible to interact with a service described by an endpoint reference (*R4*).

Cancellation is fully supported in BPEL through event handling and exception handling facilities (*R5*). Also, BPEL has built-in timing capabilities (*R6*). Timers can be attached to scopes and can be put into regular control flow, defining the maximum time that should be spent waiting. Furthermore, delays can be realized through wait activities.

BPEL is tightly integrated with WSDL, where message formats are specified and thus make BPEL supporting Requirement *T1*. The variables used at send and receive activities have to match the used operation in the expected data structure. Partner link types are configured with WSDL port types and BPEL's communication activities depend on WSDL operations. This makes BPEL rigid in terms of interchangeable technical configurations (*T4*): changes on the WSDL side often require changes in the BPEL process.

In BPEL, correlation sets are used to define which process instance an incoming message should be routed to. These correlation sets can be initialized by the BPEL engine and correlation information is included into the messages exchanged. The so called properties define where exactly in the message correlation information can be found. BPEL supports Requirement *T2*. BPEL also offers a variety of possibilities to react to erroneous situations (*T3*). Exceptions can be handled and completed activities can be compensated if necessary.

The integration between executable BPEL and abstract BPEL is already described in the specification. Although issues like conformance checking lie outside the scope of the BPEL specification, it is defined what elements must be used and what attributes must be set in either of the two BPEL flavors. As the elements and attributes available do not significantly differ from abstract BPEL to executable BPEL, we conclude that Requirement *T5* is supported.

BPEL$^{\text{light}}$ introduces the new activity type `interactionActivity`, which replaces BPEL's communication activities. Instead of using WSDL artifacts, each interaction activity is assigned to a `conversation`. To enable the usage of BPEL$^{\text{light}}$ processes in a web service environment, the concept of "assignment" is introduced. There, each `conversation` is assigned to a partner link and each activity is assigned to a WSDL operation. This makes BPEL$^{\text{light}}$ fulfill requirement *T4*.

Web Services Choreography Description Language (WS-CDL)

WS-CDL does not have a graphical notation (*C1*). As all role types are enumerated in a WS-CDL choreography and behavioral dependencies could be omitted, a structural view is present (*C2*). However, the number of services per role is unspecified. The language does not include the notion of complex interactions that can be refined (*C4*). While the concept of invoking

subchoreographies allows a certain degree of modularity, modularity and reusability across different WS-CDL choreographies is not possible. Therefore, it only partially supports modularity of choreographies (*C3*) and does not support reusability of choreographies (*C5*).

WS-CDL supports multi-lateral choreographies (*R1*). However, ownership of choice cannot be represented in WS-CDL (*R2*). As concrete services are identified by channel instances, there can be potentially many services involved in one choreography of a particular role type. However, there is no control flow construct allowing multiple branches to be executed in parallel where the number of branches is only known at runtime. Therefore, WS-CDL does not support "Multiple instances with a priori runtime knowledge" [17], nor does it fully support "One-to-many send/receive" [36], where the exact number of services might only be known at runtime. We conclude that there is only partial support for *R3*.

The notion of reference passing (*R4*) is directly integrated into WS-CDL. Only simple cases of cancellation can be represented using WS-CDL, canceling a whole choreography region through a cancellation message cannot be modeled (*R5*). Timing is represented through built-in features of WS-CDL (*R6*).

While WS-CDL focuses on the behavioral aspects of a choreography, it relies on WSDL for the specification of message formats. Therefore, it supports requirement *T1*. However, as a drawback of this integration with WSDL, the WSDL-configurations heavily influence the way the choreography looks like. Changes in WSDL files often require changes in the WS-CDL choreography. For instance, splitting a port type into several port types requires major refactoring of the choreography. Therefore, WS-CDL does not support *T4*.

Correlation of interactions is addressed using identity tokens that are included in messages (*T2*). The so called token locators are a mechanism to retrieve tokens from messages: an XPath expression defines where in the message the correlation information is placed. This is an example of how tightly WS-CDL is linked to WSDL. Whenever the message format changes, the token locators have to be adapted to the new format. This direct influence of message format changes to the choreography definition is not desired.

There is a variety of exception types available to cover message transmission exceptions, security failures and application failures (*T3*).

There have been proposals to generate abstract BPEL processes out of WS-CDL choreographies [159]. However, it remains unclear how constructs like blocking work units can be realized in BPEL. WS-CDL also allows the specification of mixed choices on a global level, i.e. choices between send and receive activities. Mixed choices on a global level are a major challenge when trying to properly translate them to interacting BPEL processes. Here, sophisticated synchronization mechanisms might need to be applied. Furthermore, round-tripping between WS-CDL and BPEL seems unrealistic given e.g. the fact that BPEL includes a `forEach` construct which does not have a correspondence in WS-CDL. Actually, the main criticism regarding WS-CDL centers around the missing alignment between WS-CDL and BPEL or other orchestration languages (*T5*).

Summary of the Evaluation

Two types of choreography languages can be distinguished: those following the *interconnection modeling style* and those following the *interaction modeling style*. Interconnection models have

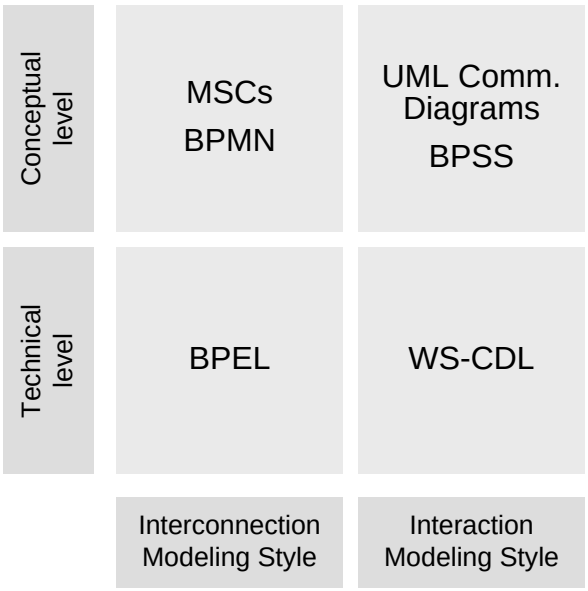


Figure 3.4: Classification of languages

communication activities as basic building blocks and behavioral dependencies are defined between them on a per-role basis. I.e. each dependency is assigned to one role only. Interaction models, in contrast, have interactions as their basic building blocks and behavioral dependencies are defined between them. This implies that the dependencies are truly global in the sense that they are not assigned to any of the roles involved.

Figure 3.4 shows a classification of the abovementioned languages. MSCs, BPMN and BPEL fall into the category of interconnection modeling. Here, message send and receive activities are distinguished. Control flow is defined on a per-role basis. In contrast to this, UML communication diagrams, BPSS and WS-CDL fall into the category of interaction modeling. Here, control flow dependencies are not assigned to individual roles. A second separation is made between languages on a conceptual level and those on a technical level.

Table 3.1 gives an overview of the assessment of the different languages. Among the languages with a graphical notation, we see that BPMN outperforms MSCs and UML Collaboration Diagrams regarding many requirements.

BPEL scores best among the technical languages. However, BPEL is not a choreography language. WS-CDL, when compared to BPEL, scores worse regarding the representation of ownership of choices, multiple participants per role, cancellation and interchangeability of technical configurations. Especially the fact that WS-CDL is not well aligned with orchestration languages leaves service choreographies and service implementation disconnected.

The assessment shows that none of the languages is perfectly suited for choreography modeling – either on the conceptual or on the technical level. Introducing a completely new language to overcome the limitations is not desirable, as this would hamper reuse of existing tools and techniques. Regarding the conceptual level, it is obvious that BPMN should be the language

Requirements	MSC	BPMN	BPSS	UML Comm. Diagrams	BPEL	WS-CDL
C1. Graphical notation	+	+	–	+/–	–	–
C2. Structural view	–	+	+	+	–	+
C3. Modularity of choreographies	–	+	–	–	+/–	+/–
C4. Decomposition of interactions	–	–	–	–	–	–
C5. Reusability of choreographies	–	–	–	–	+/–	–
R1. Multiple roles	+	+	+	+	+	+
R2. Ownership of choices	–	+	–	–	+	–
R3. Multiple participants per role	–	–	–	+/–	+	+/–
R4. Participant reference passing	–	–	–	–	+	+
R5. Cancellation	–	+	–	–	+	–
R6. Time constraints	–	+	+	–	+	+
T1. Message formats	–	+	+	–	+	+
T2. Correlation	–	–	–	–	+	+
T3. Exception handling	–	+	+	–	+	+
T4. Interchangeability of techn. configs	–	–	–	–	–	–
T5. Integration with orchestration languages	–	+/–	–	–	+	–

Table 3.1: Evaluation Summary

of choice – requiring a number of enhancements to fully support the requirements though. Regarding the technical level, the introduction of choreography extensions for BPEL, providing a structural view, a loose link between behavioral definitions and technical configurations as well as service sets as first-class citizens is the direction of choice.

3.5 Assessment of Correctness Criteria

The choreography design approaches highlighted in Section 3.1 give a good indication of the steps followed until a detailed specification is reached. Bottom-up approaches most often boil down to interconnecting existing participants and analyzing the resulting interaction behavior. The most common anomaly that can occur is a deadlock. Compatibility checking as presented in Section 2.5.1 deals with this issue.

In top-down approaches the choreography is put first and refined in a step-wise manner. Here, we need to ask the question of which choreographies can be implemented by participants. Again, the most obvious anomaly is a deadlock, the absence of which is checked through compatibility checking. Furthermore, individual interactions might not be reachable, this can be verified using standard reachability analysis e.g. for transitions in open nets.

An interesting observation regarding all choreography verification techniques is that they

tend to focus on individual conversations only. This approach is directly taken from verification of orchestrations. Due to the centralized execution and coordination of orchestrations, isolation between different process instances can easily be achieved. The engine controlling the execution can simply use a process instance identifier to distinguish the instances. In choreographies this isolation between conversations is not straightforward. The usage of a single conversation identifier is very uncommon. Rather, the participants introduce their own identifiers, requiring the other participants to include this identifier into response messages. No verification technique is available that acknowledges this fact. Therefore, extensions to existing techniques or novel techniques need to be introduced for considering multiple concurrent conversations.

Open nets are a widely used formalism for which many translation approaches from choreography languages exist, especially BPEL and BPMN. On the other hand, open nets do not include the notion of value passing and therefore only have limited suitability for distinguishing different messages of the same type. We have seen that colored Petri nets overcome this limitation. However, the syntax and semantics are very complex, making formal verification also very complex. On the other hand, π-calculus also provides intuitive support for value passing. It even incorporates the notion of fresh names, i.e. the creation of values that are not yet present in a system. This ensures uniqueness of values. In contrast to this, colored Petri nets rather operate with functional dependencies between inputs and outputs of a transition. Here, ensuring unique values is a lot harder. A hybrid approach between open nets and π-calculus seems very promising for tackling the issue of checking isolation of conversations.

While compatibility checking and instance isolation checking can ensure proper interaction between participants, these techniques do not guarantee that there exist participants that can actually collectively behave as specified in the choreography. This might be due to assumptions underlying the choreography model. This is especially true in case the choreography follows the interaction modeling style, where the specification of global control flow dependencies assumes a central observer in the first place, which does not exist in reality.

As the previous chapter pointed out, the issue of *realizability* has already been identified in the literature. Most notably, Fu et al. discussed realizability of conversation models together with a verification approach based on trace-based equivalence notions [113]. This is an important first step for tackling the realizability issue. However, it turns out that there are many subtle notions of realizability that need to be addressed. Along the three dimensions communication model, complete vs. subset of behavior and behavioral equivalence notions we can identify several flavors of realizability that need to be addressed individually. From a theoretical perspective, proper treatment of branching structures, i.e. location and moment of choice, increases the challenge.

In this discussion, not only the location of realizability problems but also resolution strategies are of interest. Here, many alternatives for repairing a choreography might be available. It is not a mere theoretical problem which alternative to choose but even more important the business impact of such a resolution decision needs to be taken into account. As we are dealing with autonomous participants, most often legally independent organizations with their own interests and goals, repairing a choreography in one direction or the other can have huge influence on the business semantics behind the choreography.

Chapter 4

Interconnection Models

The previous chapter has compared a number of prominent choreography languages and has highlighted their limitations with regards to the requirements framework. This chapter is going to tackle some of the open issues by proposing extensions for existing languages. Most parts of these languages are reused unchanged in order to ease adoption of the extensions among human modelers. By increasing the expressiveness of these languages the support for the requirements is broadened. As second major contribution of this chapter, instance isolation is introduced as novel quality criterion for choreographies.

On the conceptual level, BPMN turned out to be a promising choreography language. Deficiencies can be observed when it comes to decomposition of interactions, reusability of choreographies, representing multiple participants per role and participant reference passing. Section 4.1 introduces additional modeling constructs in order to overcome these limitations.

On the technical level, BPEL scored best regarding the requirements framework. Being a classical orchestration language, it concentrates on individual roles only. Therefore, the main challenge is to lift BPEL to a choreography language. Section 4.2 introduces BPEL4Chor as thin choreography layer on top of BPEL. BPEL4Chor was designed in close collaboration with researchers from the University of Stuttgart.

The integration between choreography and orchestration languages is of key importance in order to facilitate round-tripping between specifications and implementations. Section 4.3 investigates this issue. Mapping extended BPMN to BPEL4Chor will be discussed as well as the opposite direction of generating extended BPMN from BPEL processes.

As shown in Section 2.5 a number of verification techniques, such as compatibility and conformance checking, are already in place for interconnection models. However, the issue of proper correlation configurations was neglected in the literature. Section 4.4 presents the notion of *instance isolation* which ensures that one process instance cannot participate in multiple conversations. So called ν^*-nets are introduced as formal model for this purpose, extending open nets with the capabilities of name passing and name creation, as it is known from π-calculus.

Section 4.5 reports on practical experience with interconnection modeling and discusses a number of modeling anti-patterns.

4.1 BPMN Extensions

While BPMN already provides support for many choreography concepts, extensions are necessary to reach broader support for the requirements listed in Chapter 3. To that end, this section introduces extensions that allow the representation of multiple participants, reference passing and complex interactions.

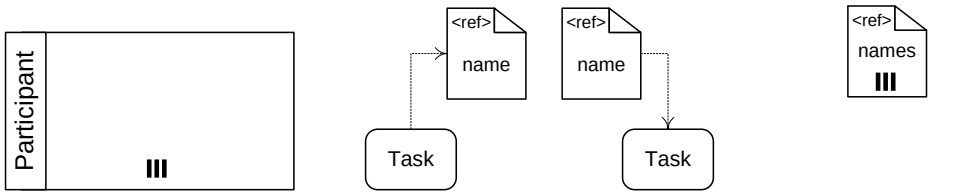
4.1.1 Extensions Overview

Participant Sets

Roles are represented as pools in BPMN. We need to distinguish those cases where at most one participant of a particular role is involved in one conversation or if there can be potentially many participants involved. In our auctioning example, there is exactly one seller and one auctioning service involved in one conversation. However, we have potentially many bidders involved.

With current BPMN, a modeler has two options. Either one pool is used and labeled "bidders" or two pools are used and labeled "bidder 1" and "bidder n". In the former case, only the label indicates that there are potentially many bidders involved in one auction. In the latter case, the behavior of role bidder needs to be duplicated and still only the label indicates that there might be many bidders. We argue that multiple participants of the same role should be a first class citizen of a choreography language, just like multiple instances activities are a first class citizen of BPMN.

For representing multiple participants we introduce pools with three bars as new notational element, shown in figure 4.1(a). A set of participants of the same type involved in the same conversation is called a *participant set*. The three bars are reused from multiple instances activities and should therefore be intuitively understood by BPMN modelers.



(a) Participant sets (b) References (c) Reference sets

Figure 4.1: Participant sets, references and reference sets

References

The main challenge with participant sets is that we need to distinguish individual participants out of this set. We do this via references as shown in figure 4.1(b). A reference is a special data object enhanced with ⟨ref⟩. A reference can be connected to a flow object via associations. We give the following semantics to the different connection directions:

- A reference can be written by a flow object (represented by an association from the flow object to the reference). (i) If the flow object is a receive activity, e.g. an intermediate message event or an activity with incoming message flow, the reference will point to the message's sender upon message receipt. If the reference already pointed to a participant, the reference will simply be overwritten. (ii) If the flow object is not a receive activity, it is not specified what participant the reference will point to. Consider the selection of the buyer in our example.

- A reference can be read by a flow object (represented by an association from the reference to the flow object). (i) If the flow object is a send activity, the message will be sent to the participant the reference points to. In our example the auctioning service sends a completion notification to exactly that bidder out of the bidder set, who was selected to have won the auction. (ii) If the flow object is a receive activity, then a message is only awaited from the defined participant. E.g. the seller only waits for payment from the buyer. (iii) If the flow object is neither a send nor a receive activity, it is not specified what happens with that reference inside the activity.

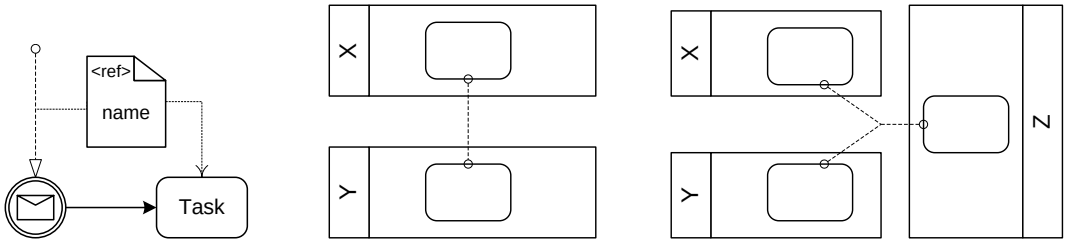
References cover those cases where an individual participant needs to be identified. However, we might need to select subsets of the participants involved in one conversation. In our example, this is the case for those bidders who did not win the auction. We need to send a sorry message to all of them—but we must not send this message to the winning bidder. We introduce reference sets as shown in figure 4.1(c) with the following semantics:

- A reference set can be modified by a flow object (represented through an association from the flow object to the reference set). (i) If the flow object is a receive activity, a reference to the sender of the message will be added to the reference set if such a reference is not already contained in the set. In our example we find a "receive bids" activity where bids from different bidders are received. In case a bidder who has already placed a bid in the same auction places another bid, no reference will be added to the set. However, if a new bidder takes part, a reference will be added. (ii) If the flow object is not a receive activity, it is not specified, what exactly happens with the set. It might be overwritten completely or references might be added, removed or changed.

- A reference set can be read by a flow object. (i) If the activity is a looped activity, i.e. a sequential loop or a multiple-instances activity, the reference set determines the number of repetitions or instances. This requires that at most one reference set serves as input for a looped activity. A special case is a looped send activity. Here, a message is sent to every of the referenced participants. In those cases, where the looped activity is a complex activity, a reference can be placed inside this activity which will represent the selected reference out of the set for a particular instance or repetition. (ii) If the activity is not a looped activity, it is not specified how the reference set is used within the activity. In our example, the "select buyer" activity takes the reference set as input and selects the winning bidder.

Reference Passing

References can be passed to other participants as shown in figure 4.2(a). The reference is connected to a message flow with an undirected association. The passed reference can be connected to other flow objects with directed associations. In figure 4.2(a), the passed reference is used in the task.



(a) Reference passing (b) Involving two roles (c) Involving three roles

Figure 4.2: Reference passing and complex message flow

Complex Message Flow

BPMN only considers interactions on the message-level, i.e. elementary interactions. In order to overcome the missing abstraction mechanism for decomposition of interactions, we introduce the complex message flow construct. The main difference to BPMN's standard message flow is that complex message flow is not directed and can involve more than two roles.

Figure 4.2(b) illustrates complex message flow between two roles (represented by dashed lines with circles at the end). The number of corresponding message flows is not specified for a complex message flow. Potentially many but at least one message will be exchanged if a complex message flow is executed. Figure 4.2(c) illustrates complex message flow between three roles. Here, again, it is left unspecified what concrete elementary interactions between the roles apply.

4.1.2 Example

Figure 4.3 shows an example including some of the proposed extensions. First of all, three bars were added to the pool of the bidder to represent a participant set.

The "receive bid" task of the auctioning service collects the references of the different bidders into a reference set. The reference set is forwarded to the "select buyer" task. Inside this task, the successful bidder is selected and placed into a new reference, denoted as buyer. The remaining references of the bidders reference set are placed into the "others" reference set. The others reference set is used as an input to the "send sorry message" task. Here, an instance is created for each element of the set. Hence, all unsuccessful bidders are notified.

The buyer reference is forwarded to the "send completion notification" task, where it determines the instance of the bidder that should be contacted. Furthermore, it is passed to the seller, where it is used as an input for the reception of the payment as well as determining the reference of the bidder's instance to which the product should be sent.

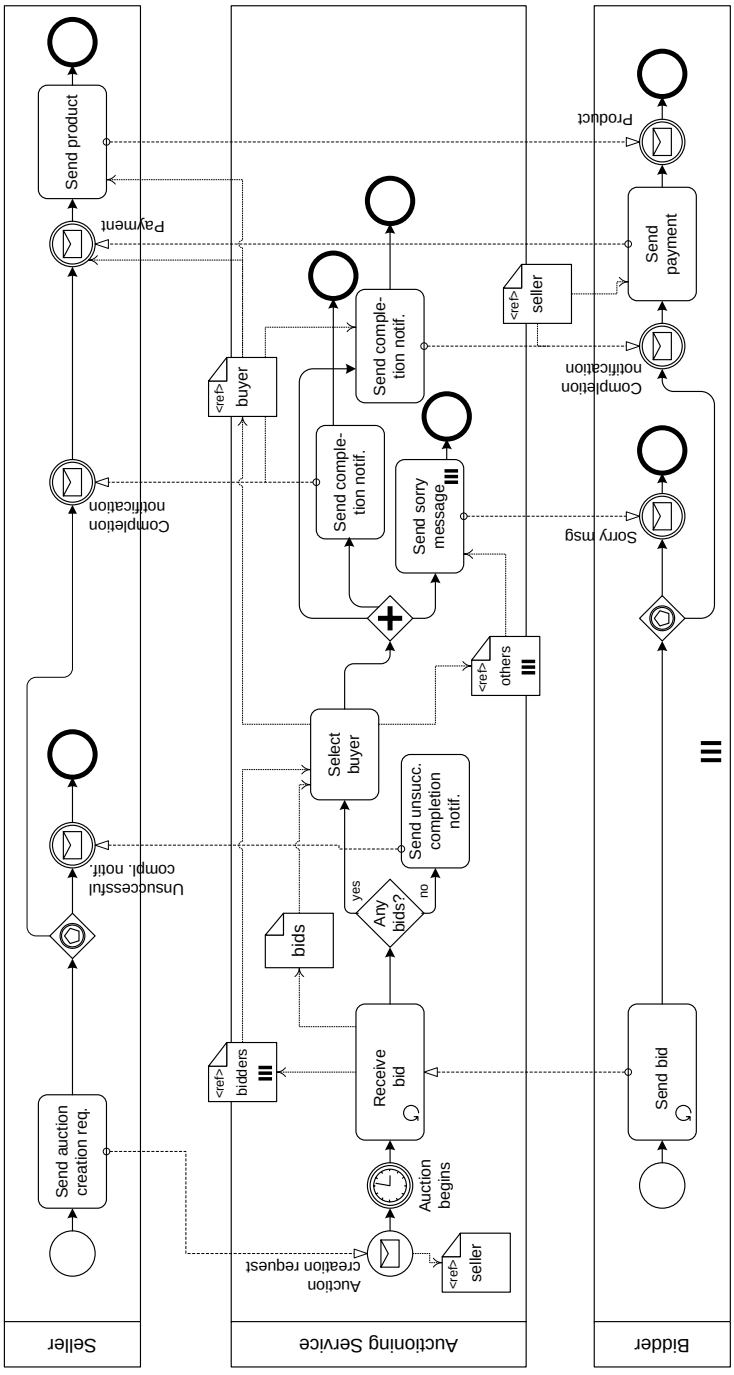
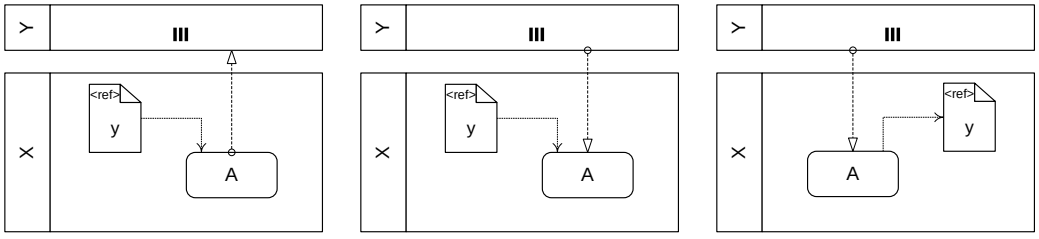


Figure 4.3: The auctioning scenario represented using extended BPMN

Finally, a reference of the seller is passed to the successful bidder. This reference is acquired via the initial interaction between the seller and the auctioning service. Two data objects are used for this purpose where both carry the same label. According to standard BPMN semantics, both elements refer to the same data object at runtime.

4.1.3 Validation

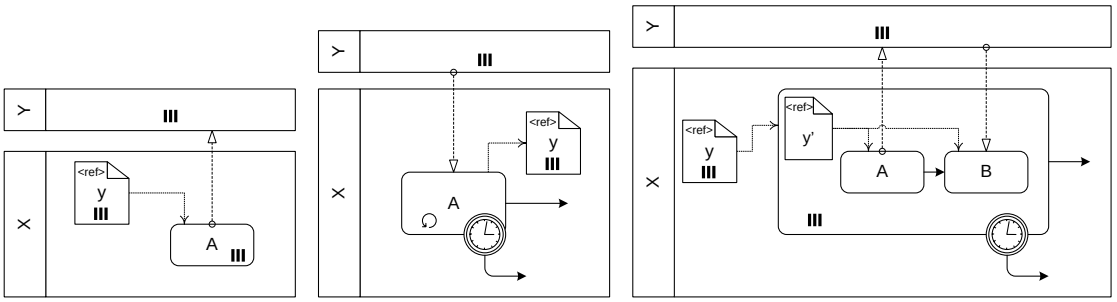
The proposed extensions broaden BPMN's support for the Service Interaction Patterns [36]. Especially the additional capabilities of references and participant sets allows to describe scenarios with potentially many participants of the same role in the same conversation (requirement *R3*). Figure 4.4 illustrates the patterns Send, Receive and Send/Receive.



(a) Send to Reference (b) Receive from Reference (c) Receive Reference

Figure 4.4: Single transmission bilateral interaction patterns

In the Receive pattern, the receiver automatically gains knowledge about the reference of the requester. If the message should be received from a particular instance of a participant set, a reference according to figure 4.4(b) has to be used. If a message is received from an unspecified instance of the participant set, the corresponding reference can be collected, shown in figure 4.4(c). The representation of the single transmission multilateral interaction patterns are



(a) One-to-many send (b) One-from-many receive (c) One-to-many send/receive

Figure 4.5: Single transmission multilateral interaction patterns

equally influenced by the novel constructs participant sets and references (cf. Figure 4.5).

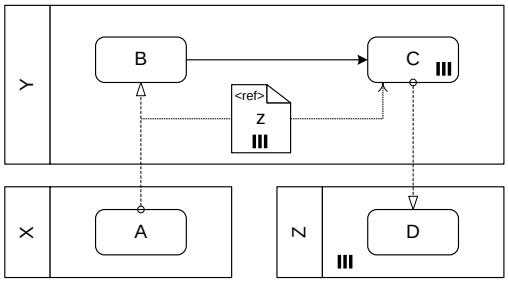


Figure 4.6: Request with Referral

Reference passing (Requirement *R4*), as present in the Request with referral pattern, can be achieved using standard data flow mechanisms. The additional typing of data objects in the direction of references makes this possible. The realization of the pattern is shown in Figure 4.6. The instances of Z that should receive the follow-up responses are passed on using the reference set.

Correlation (*T2*) and interchangeability of technical configurations (*T4*) were not addressed. These are requirements for choreography languages on the technical level and therefore do not apply to the extended BPMN, as it operates on the conceptual level.

Another drawback of plain BPMN is the missing support for decomposition of interactions (*C4*). Although the extensions come with a complex message flow construct, the relationship between such a complex message flow and elementary interactions, i.e. regular message flow, cannot be made graphically. Reusability of subchoreographies (*C5*) was not addressed, either.

4.1.4 Discussion

The proposal makes heavy use of refined data objects. A major problem with BPMN data objects is that their semantics is not clearly defined in the BPMN specification. E.g. it is unclear what it means if different activities write on the same data object. Here, we simply assume that if an activity has write-access to a data object, it (might) overwrite the entire content of the data object upon completion. BPMN does not have the notion of collections or buffers, as they are present in UML Activity Diagrams [5]. Therefore, we introduced a distinction between simple data objects and data object sets, where we assume that write-access to a data object set typically means that the activity (might) add an object to the set. We do not require that data objects are only placed within pools or only accessed from within one pool.

References express correlation in those cases where receive activities read references. This defines whom messages are to be received from. However, this notion of correlation only covers a limited set of scenarios. Imagine settings, where the same pair of participants engage in different concurrent conversations. Here, our notion of references is not sufficient to distinguish the different conversations. Furthermore, it might be important to specify what message parts correlation is actually based on. E.g. a customer id or a shipment invoice number might be used as concrete correlation identifiers. There might be even more sophisticated correlation mechanisms needed, such as ranges of values or time-based correlation of messages. [35] provides a set of correlation patterns that could serve as starting point for further refinements regarding

correlation support.

While complex message flow allows to model interactions on different levels of abstraction, a clean decomposition mechanism is still not present. This is caused by the the fact that message flow is represented using lines in BPMN. Decomposition of lines is hard to integrate into a visual notation. As a direct implication, integrating reusability into BPMN is not easy. Without a clean decomposition concept, reusability cannot be realized. Heavy use of textual attributes could have been chosen as workaround. In order to keep a clean language targeted at a conceptual level, we decided not to integrate this.

4.2 BPEL4Chor

BPEL already supports a wide range of requirements from Chapter 3. The main issue is that BPEL is an orchestration language and not a choreography language. The specification of observable behavior models is already supported through abstract BPEL. In order to allow choreography modeling, it must be possible to describe the behavior of multiple services.

The second challenge that needs to be overcome is that of interchangeability of technical configurations. The tight integration between BPEL and WSDL needs to be relaxed as demanded by requirement *T4*. Therefore, BPEL4Chor decouples the "heart" of choreographies, i.e. the communication activities, their behavioral dependencies and their interconnection, from technical configuration, e.g. the definition of WSDL port types. That way, a higher degree of reusability of the choreography models is achieved.

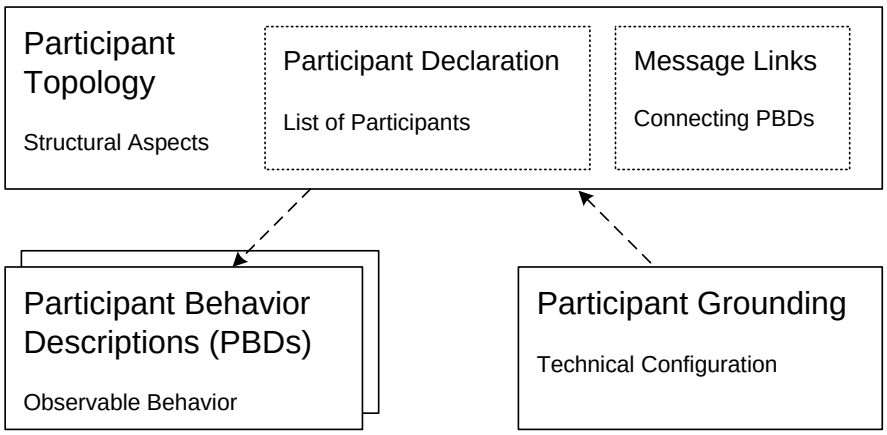


Figure 4.7: BPEL4Chor artifacts

BPEL4Chor is a collection of three different artifact types. Each artifact type is represented as a rectangle in Fig. 4.7 and the dashed arrows between the rectangles symbolize that there exist references from artifacts of one type to artifacts of the other type.

1. A *participant topology* defines the structural aspects of a choreography by specifying participant types, participant references, and message links.

2. *Participant behavior descriptions* define the control flow and data flow dependencies between activities, in particular between communication activities, at a given participant. By assigning a message link to a communication activity, participant behavior descriptions plug into the topology.

3. *Participant groundings* define the actual technical configuration of the choreography. Here, the choreography becomes web-service-specific and the link to WSDL definitions and XSD types is established. We use the term "grounding" similar to the Semantic Web terminology, where "grounding" denotes that the semantically described service is bound to a concrete technical configuration.

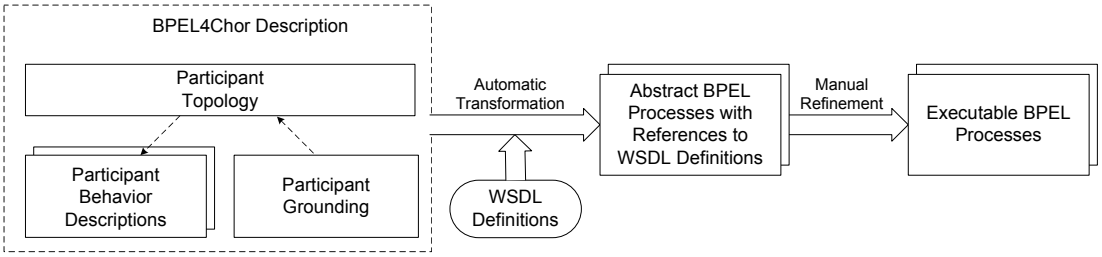


Figure 4.8: Getting from a BPEL4Chor choreography to executable BPEL processes

It is possible to get an executable process from each participant behavior description as shown in Figure 4.8. First of all, all technical configurations of each participant have to be specified in the participant grounding. This serves as input for a transformation that can be run. This transformation transforms each participant behavior description into a BPEL process, where the technical details given in the participant grounding are included. However, internal activities, such as data manipulation or calling other services are missing. These have to be added manually afterwards. The details of the transformation itself are described in Section 4.2.6.

The following sections are going to introduce the artifact types of BPEL4Chor. Corresponding code snippets are given for the example from Section 1.1.

4.2.1 Participant Topology

The participant topology describes the structural aspects of a choreography. As most important aspect, the participant topology enumerates different *participant types*. BPEL4Chor supports the following three cases: (i) there is only one participant of a certain type in one conversation. As an example, an individual seller and a single auctioning service involved in a specific auction can be imagined. (ii) Several participants of a certain type appear in one conversation and the number of participants is known at design-time. As an example imagine a scenario where two shippers are involved in one conversation. (iii) An unbounded number of participants is involved and the exact number might only be determined at runtime. Imagine a large number of bidders involved in our sample scenario.

Participant types are not sufficient to support cases (ii) and (iii), as we need to distinguish between different participants of the same type. E.g. we need to distinguish between a bidder

who has won the auction from a bidder who has not. Therefore, the notions of *participant reference* and *participant set* are introduced. A participant reference describes an instance of a participant type, e.g. one particular bidder, while a participant set describes a set of participant references, e.g. the set of all unsuccessful bidders.

Cardinality of participant types is implicitly given through the participant reference and participant set declarations. (i) If there is only one participant reference for a given participant type, we can conclude that there will be at most one participant of that type involved in a conversation. (ii) If there are several participant references but no participant set for a given type, then the maximum number of participants in one conversation is limited by the number of references. (iii) If there is a participant set declared for a given type, then the number of participants is not defined at design-time.

```
<topology name="topology" targetNamespace="urn:auction"
    xmlns:sns="urn:auction:seller">
 <participantTypes>
  <participantType name="Seller" participantBehaviorDescription="sns:seller"
      />
  <participantType name="AuctioningService" ... />
  <participantType name="Bidder" ... />
 </participantTypes>
 <participants>
  <participant name="seller" type="Seller" selects="auctioningService" />
  <participant name="auctioningService" type="AuctioningService" />
  <participantSet name="bidders" type="Bidder">
   <participant name="bidder" selects="auctioningService" />
   <participant name="successfulBidder" />
  </participantSet>
  <participantSet name="unsuccessfulBidders" type="Bidder"
   forEach="as:notifyUnsuccesfulBidders">
   <participant name="currentBidder" forEach="as:notifyUnsuccesfulBidders"
       />
  </participantSet>
 </participants>...
</topology>
```

Listing 4: Participants in the participant topology

Listing 4 shows the participants in the participant topology for the auctioning scenario. Three participant types, namely seller, auctioning service and bidder, are listed along with a participant reference for a seller and an auctioning service, respectively. A participant set for the bidders is also given. The participant reference `currentBidder` contained in the set will be used as iterator on the set later on. `successfulBidder` is one particular participant from the set of bidders. In general, the semantics of a reference p contained in a set s is that if a sender p is not contained in s, then this new sender is added to the set. Using different participant references does not guarantee that the referenced participants are different at runtime. The same applies to sets, which may overlap in terms of referenced participants.

In Listing 4, the declaration of the seller declares a `selects` attribute that refers to the auctioning service. This indicates that the seller chooses which auctioning service she actually

wants to use. This in turn implies that there are potentially many auctioning services available. The selection of participants might happen at runtime or already at design-time. The topology in Listing 4 does not exclude the case that every seller has exactly one auctioning service she always goes to.

The attribute `forEach` on the set of unsuccessful bidders denotes that the `forEach` activity having the name `notifyUnsuccesfulBidders` at the auctioning service should iterate over that set. The attribute `forEach` at the nested participant `currentBidder` denotes that this participant reference should store the current value of the iterator. A `forEach` may only iterate on one set. Thus, we require that for each `forEach` there is at most one participant set and at most one participant reference pointing to it.

```
<topology name="topology" targetNamespace="urn:auction" ...> ...
 <messageLinks>
  <messageLink name="auctionRequestLink"
   sender="seller" sendActivity="sendAuctionCreationRequest"
   bindSenderTo="seller"
   receiver="auctioningService"
   receiveActivity="receiveAuctionCreationRequest"
   messageName="auctionCreationRequest" />
  ...
  <messageLink name="bidLink"
   senders="bidders" sendActivity="sendBid"
   bindSenderTo="bidder"
   receiver="auctioningService" receiveActivity="receiveBid"
   messageName="bid" />
  <messageLink name="bidAckLink"
   sender="auctioningService" sendActivity="sendBidAck"
   receiver="bidder" receiveActivity="receiveBidAck" />
  ...
  <messageLink name="completionNotificationLink"
   sender="auctioningService" sendActivity="sendCompletionNotification"
   receiver="seller" receiveActivity="completionNotification"
   messageName="notification"
   participantRefs="successfulBidder" />
  <messageLink name="unsuccessfulBidLink">
   sender="auctioningService" sendActivity="sendUnsuccessfulBid"
   receiver="currentBidder" receiveActivity="receiveUnsuccessfulBid"
   messageName="notification" />
  </messageLink> ...
 </messageLinks>
</topology>
```

Listing 5: Message links in the participant topology

Listing 5 shows the message links in the participant topology for the auctioning scenario. *Message links* state which participant can potentially communicate with which other participants. However, the topology does not include any constraint about ordering sequences or the cardinality of message exchanges. BPEL4Chor is based on a *closed world assumption*. In particular, message links only connect participants listed in the topology. I.e. the sender and receiver attributes must always be set.

Some of the message links in Listing 5 also contain the attribute `participantRefs`. This attribute realizes link passing mobility in BPEL4Chor: as part of the exchanged business documents, participant references are passed from one participant to another. E.g. consider the message link `completionNotificationLink`. Here, the reference to the successful bidder is passed from the auctioning service to the seller. This enables the seller to directly communicate with this bidder later on.

Reference passing must also happen whenever the attribute `bindSenderTo` is set for a message link. In contrast to `participantRefs`, where a reference to a third participant is passed, `bindSenderTo` implies that the sender of the message must include a reference to herself in the message. As knowledge about participants is local, the usage of `bindSenderTo` is required even in those scenarios where only one participant of a type is involved. Take `auctionRequestLink`. Here, the seller sends a reference pointing to herself, enabling the auctioning service to reply later on.

Selection and reference passing lead to the *binding* of concrete participants to participant references. In the case of reference passing, rebinding occurs if a participant was already bound to a reference. The reference is simply overwritten. If a referenced participant should be bound to a different participant reference, the attribute `copyParticipantRefsTo` is used.

Special semantics applies if binding occurs for a participant reference contained in a participant set. Here, the reference must be added to the set upon binding, provided that a reference to that participant is not yet contained in the set.

The listing also shows that either the attribute `sender` or `senders` is used in a message link. `sender` is used if one participant reference applies, `senders` is used if any participant out of a set can be the sender.

Topologies by themselves only describe the structural aspects of a choreography. However, the typical usage for topologies is to glue together participant behavior descriptions. Therefore, one already finds connections to constructs from participant behavior descriptions in the topology. These attributes are explained in the next section.

4.2.2 Participant Behavior Descriptions

Participant behavior descriptions (PBDs) cover the behavioral aspects of a choreography. Control flow dependencies between communication activities are defined per participant type. These dependencies determine the ordering sequences of message exchanges the different participants have to adhere to. Furthermore, participant behavior descriptions also cover data flow aspects. It is important to specify what data can be expected by the receiver of a message and how this data relates to data previously exchanged.

Abstract BPEL is used as basis for participant behavior descriptions. It already provides most constructs that are needed. In contrast to executable BPEL, some language constructs do not need to occur and some attributes do not need to be set. BPEL profiles force or forbid the usage of certain attributes in abstract BPEL process. Therefore, we introduce the *Abstract Process Profile for Participant Behavior Descriptions* stating the requirements for the definition of the behavior of one participant. This profile inherits all constraints of the *Abstract Process Profile for Observable Behavior* from the BPEL specification. This allows to add e.g. opaque activities into a PBD, which is useful for documentation purposes.

Communication activities Message send and receive activities are at the center of attention in choreographies. BPEL uses `invoke` and `reply` as send activities and `receive` and `onMessage` as receive activities. These activities are reused in BPEL4Chor.

The usage of `partnerLink`, `portType` and `operation` attributes at the communication activities of BPEL link the BPEL process tightly to WSDL operations. The communication activities are linked together using message links in the topology as described in the last section. Thus, a linkage to WSDL artifacts is obsolete. Therefore, we forbid the usage of the attributes `partnerLink`, `portType` and `operation` at communication activities.

In the topology, a message link references two communication activities. To enable proper referencing, we need an identifier for each communication activity in a participant behavior description. Since `onMessage` branches do not offer an attribute `name`, we introduce the attribute `wsu:id` having the type `xsd:id` as new attribute for communication activities and `onMessage` branches. To simplify reading, the `wsu:id` attribute defaults to the attribute `name` of the BPEL activity. Typically, one message link references exactly one send and one receive activity. However, there are scenarios, where one message link references several send and/or receive activities. E.g. several send activities can have the same target activity or several receive activities refer to the same message type.

BPEL directly adopts some of the WSDL interaction styles. BPEL distinguishes between one-way interaction and request/response interactions. We argue that the choice of the interaction style is a configuration issue and should normally happen in the phase of the technical configuration. Nevertheless, we allow that both interaction styles can be used in the participant behavior descriptions. This allows higher similarity between participant behavior descriptions and orchestrations. Especially in a bottom-up approach where existing BPEL files are the starting point, we do not want to force the modeler having to split e.g. a request/response activity into two activities in the BPEL code. In this context, we force the attribute `messageExchange` to be present in order to relate pairs of `receive` and `reply` activities. This constraint is necessary, because corresponding pairs can no longer be determined by matching `portType` and `operation` values.

Control flow BPEL comes with a rich set of control flow constructs, which are used unchanged in BPEL4Chor. This enables the reuse of existing BPEL tools to model choreographies. As examples `if`, `pick`, `flow` and `forEach` are available to model branching structures and concurrency. BPEL's facilities to model time constraints, namely `wait` and event handlers with `onAlarm`, are also reused unchanged.

Support for participant sets Scenarios where a number of participants of the same type are involved in one conversation are recurrent in the choreography world. BPEL supports parallel instances the number of which might only be known at runtime through the `forEach` construct. Therefore, there is sufficient support for participant sets from the control flow side. However, data sets are not natively supported in BPEL. Special XSD types have to be used for this purpose. As a realization for the iteration on participant sets, the attribute `forEach` can be set for participant references and participant sets in the participant topology. The semantics is that the corresponding set determines the number of branches spawned and the participant reference is

used as iterator.

Data flow It should be specified in a choreography what contents can be expected by the receiver of a message. As an example the quantity of ordered products in an order message and the quantity in an order acknowledgment message must not differ. Furthermore, data values can determine branching behavior in choreographies. E.g. a seller might need to be ready to provide further details when the goods offered have a certain value.

BPEL offers a rich set of constructs to model data access and data manipulation. Variables are used as output of receive activities and input of send activities. Data "flows" from one activity to another by using one variable as output variable for the first activity and input variable of the other, provided that no other activity overwrites the variable in between.

In the general case, it should be possible to leave variables untyped to offer a greater flexibility to the business user. This is in line with the *Abstract Process Profile for Observable Behavior*, which does not require the specification of a type for each variable. However, typed variables are necessary to define data-based decisions in detail. Otherwise, branching conditions can only be formulated as plain text.

For documentation purposes it is sometimes useful to add hints about dependencies between data values. E.g. it could be specified that the decision about who is the successful bidder in an auction should be based on the height of the bid. Assign activities with opaque `from` or `to` parts can be used in this context.

```
<process name="auctioningService"
    targetNamespace="urn:auction:auctioningService"
  abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12">
<sequence>
  <receive name="receiveAuctionCreationRequest" createInstance="yes" /> ...
  <scope>
    <eventHandlers><onAlarm .../></eventHandlers>
    <sequence>
      <receive name="receiveBid" />
      <invoke name="sendBidAck" />
    </sequence>
  </scope>
  <flow>
    <invoke name="sendCompletionNotification" />
    <forEach name="notifyUnsuccesfulBidders">
      <scope><invoke name="sendUnsuccessfulBid" /></scope>
    </forEach>
    <invoke name="sendSuccessfulBid" />
  </flow>
</sequence></process>
```

Listing 6: Participant behavior description for the auctioning service

Message correlation BPEL comes with a built-in handling of message correlation. Since BPEL4Chor choreographies depend on BPEL and should not introduce any implementation dependencies, the correlation mechanism of BPEL is used unchanged: correlation may be specified

in the participant behavior description. We allow the usage of the attribute `correlationSet`, but use the QNames of the properties specified for a correlation set as names. Thus, the names of properties change to NCNames and therefore have no connection to property aliases. Hence, in contrast to BPEL, properties can be left untyped and are not bound to WSDL. Actual typing will happen in the participant grounding.

Listing 6 shows the participant behavior description (PBD) for the auctioning service in the example given in Section 1.1. The abstract BPEL profile for participant behavior descriptions is referenced.

4.2.3 Participant Grounding

While the participant topology and the participant behavior descriptions are free of technical configuration details, the participant grounding introduces the mapping to web-service-specific configurations. So far, port types and operations are left out and XML schema types for messages are not mandatory. In the participant grounding, these aspects are brought in. The participant grounding is specific to the target platform. While we use BPEL as target platform, it is possible to replace the participant grounding by a participant grounding specific to other target platforms, such as BPEL4SWS [161], to enable a semantic-based execution of each participant.

After a process is grounded, the participant behavior descriptions can be transformed to abstract BPEL processes, where partner links, port types and operations are defined (cf. Section 4.2.6). Since BPEL depends on WSDL 1.1 [56], message links, participant references and properties have to be assigned to WSDL artifacts in a participant grounding.

Message links are targeted at one or more receive activities. A message link models the sending and consumption of one message and does not model multicast. Therefore, one message link is grounded to one WSDL operation. This allows for realizing one participant through different port types. The attributes `participantRefs` and `bindSenderTo` enable link passing mobility in BPEL4Chor choreographies. In the case of executable BPEL, service references are passed in messages. The mapping of participant references to concrete service references is done by grounding a participant reference to a WSDL property. A WSDL property states where a certain element is located in different message types. Using that property, a BPEL process can extract the concrete service reference out of an incoming message regardless of the type of the incoming message. In that way, the service reference of a passed participant reference can be located in different messages. For correlation, we allow untyped correlation sets. With the participant grounding, these sets get typed.

Listing 7 presents the participant grounding for the auctioning example. Each message link is grounded to a WSDL operation and each participant reference is grounded to a WSDL property.

4.2.4 Consistency between BPEL4Chor Artifacts

This section will introduce a number of constraints. These constraints are used to ensure consistency between BPEL4Chor artifacts.

```
<grounding topology="auc:topology" xmlns:top="urn:auction" ...>
  <messageLinks>
    <messageLink name="auctionRequestLink"
     portType="auc:auction_pt" operation="auctionRequest" />
    <messageLink name="bidLink"
     portType="auc:auction_pt" operation="bid" />
    <messageLink name="bidAckLink"
     portType="bid:bidder_pt" operation="bidAck" />
    ...
  </messageLinks>
  <participantRefs>
    <participantRef name="seller" WSDLproperty="msgs:sellerProp" />
    <participantRef name="bidder" WSDLproperty="msgs:bidderProp" />
  </participantRefs>
</grounding>
```

Listing 7: Participant grounding

Participant references and message links If there are several senders in a message link and the attribute `bindSenderTo` is set, all senders must be of the same type as the participant reference defined in the attribute. If the attribute `copyParticipantRefsTo` is set, the list must match the `participantRefs` list in terms of length, participant types and cardinality. A participant set having an attribute `forEach` must contain exactly one participant with a matching attribute `forEach` for every `forEach` listed.

Message links and communication activities For every `invoke` and `reply` (`receive` and `onMessage`) activity there must be at least one message link in which this activity is a send (receive) activity. In the other direction, the send (receive) activities given in a message link must be `invoke` or `reply` (`receive`, `onMessage` or `invoke`) activities. If `senders` is specified in a message link l and the receiving activity is connected to a reply activity through an attribute `messageExchange`, `bindSenderTo` has to be specified in the link l.

Synchronism issues If the output variable is specified for an `invoke` activity, it must appear as `receiveActivity` in a message link. The synchronous call must be matched on the receiving side by `receive` activities with corresponding `reply` activities. On the other hand, each pair of `receive` / `reply` activities must be matched by a corresponding `invoke` activity.

Completeness of participant grounding All passed participant references, used message properties and message links must be grounded. If variable types are defined for a send or receive activity, the variable type must match the type of the expected port type. Furthermore, references passed via a message link must be grounded in a WSDL property. This applies to the two attributes `participantRefs` and `bindSenderTo`. Conflicting groundings of message links can occur especially in choreographies with synchronous interactions. The corresponding message links must be grounded in one WSDL request-response operation.

Further consistency issues The consistency constraints mentioned above can be detected easily in a BPEL4Chor choreography on a syntactical level. However, anomalies such as behavioral incompatibility require behavioral analysis. An approach for checking the absence of deadlocks and proper termination of BPEL4Chor choreographies was presented by Lohmann et al. [150], where the participant behavior descriptions are translated to Petri nets [148] and connected via communication places as it can be derived from the topology. This approach abstracts from data flow. Data-based decisions are treated as non-deterministic choices.

Another challenge is the verification of proper reference passing. In scenarios where dynamic binding is used, it must be ensured that the selection of participants is properly propagated to the corresponding participants. This aspect will be investigated further in Section 4.4.

When multiple participants of the same type are involved, it may occur that some of the combinations of communication activities and participant references are not used in message links. For example, the receive activity `receiveUnsuccessfulBid` is contained in the participant behavior description for the participant type `Bidder`. Concrete participants for this type are `bidder`, `successfulBidder` and `currentBidder`. The message link `unsuccessfulBidLink` is only one message link targeting `receiveUnsuccessfulBid`. This message link specifies `currentBidder` as receiver. Thus, the activity `receiveUnsuccessfulBid` of the participants `successfulBidder` and `currentBidder` will never be used. In the example process, this does not cause a problem, since this activity is only reached for unsuccessful bidders. In order to decide reachability for particular participants, the overall behavior has to be considered.

4.2.5 Validation

We can use the requirements from Chapter 3 to assess BPEL4Chor. This section concentrates on BPEL's white spots and how BPEL4Chor addresses them.

The main issue regarding BPEL was that it is not a choreography language. BPEL4Chor overcomes this limitation by introducing the topology as additional artifact interrelating the participant behavior descriptions which in turn are given as BPEL processes. At the same time the topology serves as structural view of the choreography (Requirement *C2*).

The usability as choreography language on the technical level can be traced back to a number of improvements over BPEL. Participant sets were introduced as first-class citizen in BPEL4Chor, allowing for easily expressing situations with potentially many participants of the same role involved in the same conversation. We can take the realization of the Service Interaction Patterns One-to-many-send and One-from-many-receive as examples.

Listings 8 and 9 show the topology and the participant behavior description for the sender role, jointly realizing One-to-many-send. A sender s sends a message to a number of recipients. The participant set `receivers` is used in combination with the `<forEach>` construct. The number of recipients does not need to be known at design-time. The sender is responsible for selecting the recipients. This obligation is specified using the `selects` attribute at the declaration of the sender `s`.

The realization of the One-from-many-receive pattern resorts to a `<while>` construct, the repetition in which the messages are collected (cf. Listing 10 and 11). The participant set `senders` represents the set of all possible senders. The second set `mySenders` contains all

```
<participants>
  <participant name="s" type="Sender" selects="receivers" />
  <participantSet name="receivers" type="Receiver" forEach="s:fe1">
    <participant name="r" forEach="s:fe1" />
  </participantSet>
</participants>
<messageLinks>
  <messageLink sender="s" sendActivity="sendDocument" receiver="r"
     receiveActivity="receiveDocument" messageName="document" />
</messageLinks>
```

Listing 8: Participant topology for the One-to-many send pattern

```
<forEach name="fe1" parallel="yes"><scope>
  <invoke name="sendDocument" />
</scope></forEach>
```

Listing 9: Participant behavior description for participant type Sender

participants whose messages are actually received. Within the `<while>` structure we find a `scope` which the participant reference `s` is limited to. This means that every time the scope is entered, no participant is bound to `s` and a message from any sender can be received. The containment relationship between `s` and `mySenders` has the semantics that if a sender not contained in `mySenders` is bound to `s`, then this new sender is added to the set. This particular semantics of BPEL4Chor participant sets makes it very easy to express such scenarios.

```
<participants>
  <participant name="r" type="Receiver" />
  <participantSet name="senders" type="Sender" />
  <participantSet name="mySenders" type="Sender">
    <participant name="s" scope="rcvScope" />
  </participantSet>
</participants>
<messageLinks>
  <messageLink senders="senders" sendActivity="sendDoc"
     bindSenderTo="s" receiver="r" receiveActivity="receiveDoc"
     messageName="document" />
</messageLinks>
```

Listing 10: Participant topology for the One-from-many receive pattern

```
<while><condition />
  <scope name="rcvScope"><receive name="receiveDoc" /></scope>
</while>
```

Listing 11: Participant behavior description for participant type Receiver

Along with the capability of selecting participants, the concrete participants involved in a conversation are not necessarily known to all participants right from the start of the conversation. Therefore, the notion of link passing mobility plays a vital role in many scenarios. It also

occurs in the Request with referral pattern. Introducing the `participantRefs` attribute of the `<messageLink>` element, BPEL4Chor makes it even easier than BPEL to express link passing mobility. Since participant sets can also be used as values for that attribute, the number of passed references does not need to be known at design-time in BPEL4Chor. Listing 12 shows the topology.

```
<participants>
  <participant name="a" type="A" selects="b c"/>
  <participant name="b" type="B" />
  <participant name="c" type="C"/>
</participants>
<messageLinks>
  <messageLink sender="a" sendActivity="sendMsg1" receiver="b"
     receiveActivity="receiveMsg1" messageName="msg1"
     participantRefs="c" />
  <messageLink sender="b" sendActivity="sendMsg2" receiver="c"
     receiveActivity="receiveMsg2" messageName="msg2" />
</messageLinks>
```

Listing 12: Participant topology for the Request with referral pattern

Another weakness of BPEL regarding the requirements framework was the inability to interchange technical configurations. This issue was resolved by introducing the notion of technical groundings, containing all the technical details such as port type and operation configurations and concrete data types.

The integration with BPEL was not hampered in BPEL4Chor. The following section will show how executable BPEL can be generated out of fully grounded BPEL4Chor.

4.2.6 From BPEL4Chor to Executable BPEL

While choreographies serve as interaction contract between business partners, they are not meant to be executed by themselves. However, they can serve as blueprint for actual business process implementation for the different partners. As the usage of executable BPEL for process implementation is a typical case, we discuss the transformation of BPEL4Chor to executable BPEL in this section. An overview of the transformation has been given at the beginning of this section. Recall BPEL offers both, the definition of abstract BPEL processes and executable BPEL processes. Executable BPEL are processes which can be deployed and run in a BPEL engine. The intended usage of abstract processes is defined by a so-called "profile". The BPEL specification defines a profile for process templates and a profile for observable behavior. The profile for observable behavior ensures that the interactions between the participants will not be changed during an *executable completion* of an abstract BPEL process. "Executable completion" is defined in the BPEL specification [108] and describes constraints on the manual actions taken to advance from an abstract BPEL process to an executable BPEL process.

When advancing from BPEL4Chor to executable BPEL two typical scenarios can be distinguished. (i) A BPEL4Chor choreography was set up and agreed upon by a set of business partners. This choreography not only includes the topology and the participant behavior descriptions but also a *complete participant grounding*. A complete participant grounding includes port types

and operations that are defined for all message links. Imagine there is one auctioning service dictating the participant grounding. Now each partner uses his generated abstract BPEL process as starting point for internal refinement. (ii) A BPEL4Chor choreography is set up. This time, the choreography is used with different participant groundings. For example, there still is only one auctioning service but some of the very important sellers require *different participant groundings*. The internal process of the auctioning service stays unchanged for different sellers. In this case, the auctioning service refines the BPEL4Chor choreography by adding internal activities and variables. Then an executable BPEL process is generated for each participant grounding and directly fed into an execution engine.

In both scenarios there is a transformation step, where—combined with the topology and participant grounding—a participant behavior description is mapped to a BPEL process following the *Abstract Process Profile for Observable Behavior* (observable BPEL for short). The transformation from a participant behavior description to observable BPEL processes is an automatic step. While most aspects are copied unchanged from the participant behavior description to the observable BPEL process, four challenges are to be tackled:

- *Generation of partner link types and partner links.* Partner links are excluded in BPEL4Chor, but required in the observable BPEL to specify the partner to interact with. Therefore, partner link types and partner links have to be generated.

- *Realization of BPEL4Chor binding semantics.* Participants are bound if their reference is passed over a message link. The pendant for participant references are service references in BPEL. Thus, participant references have to be mapped to service references.

- *Realization of participant sets.* A participant set contains multiple participant references. Since the BPEL specification is not aware of a set of service references, we have to introduce them.

- *Inclusion of participant grounding details.* A participant behavior description leaves out port types and operations. If it comes to the BPEL processes following the "Abstract Process Profile for Observable Behavior", these technical details have to be put into communication constructs, such as `invoke` or `receive`.

Generation of Partner Link Types and Partner Links

BPEL uses the notion of partner link types to connect two services. A partner link is an instance of a partner link type and is declared in a BPEL process. A partner link denotes which port type is used to send a message and which port type is offered to receive a message. The semantics of partner links is that a partner link holds service references at runtime and denotes the port type to be used. The BPEL specification does not introduce any other runtime semantics at the atomic service level. The concept of partner links is specified in the BPEL specification and added as an extension to WSDL. At the participant grounding, we use WSDL without BPEL specific extensions. Therefore, we generate partner links out of the port types given in the participant grounding.

There may be more than one partner link for a participant if the participant is realized by multiple port types: for example, the participant grounding of the auctioning scenario (Listing 7) shows two different port types for two message links targeting at the bidder.

In general, for each participant reference and message link, a partner link type and a partner link are generated. A partner link is reused, if it is visible at the current activity and if the message link belongs to the same participant reference and the message link is grounded to the same port type. The grounded port type is set to `myRole` if the message link is inbound and the port type is put as `partnerRole` if the message link is outbound.

Realization of BPEL4Chor Binding Semantics

In the context of BPEL, partner links and correlation sets are the artifacts used to realize binding semantics: if a service reference is copied to a partner link, the participant belonging to the port type is bound. As soon as a reference is passed over a message link, the reference is copied to the partner link. In the case of correlation sets, the correlation set has to be defined and the content of the message has to match it. As a next step, the message is passed to the receiving activity. The port to which the message is sent to may be bound at deployment time or at runtime. At runtime, the port may be bound at process instantiation or at the first usage of the port. If correlation sets are used to realize BPEL4Chor binding semantics, the implementer has to ensure that the correlation sets are properly used. The last moment when this may happen is when an executable completion of the abstract BPEL process is made.

In the presented auctioning scenario, references are passed over the message link, since arbitrary sellers can register at the auctioning service and the bidders are unknown at modeling time. The seller has to send his service reference to the auctioning service, which in turn uses this service reference to communicate with the seller. The message link `auctionRequest` specifies `bindSenderTo`. Therefore, the auctioning service can use the grounded WSDL property to fetch the service reference out of the received message. That service reference is then copied to the partner link used to send messages to the seller. If there are multiple partner links used for the seller, the service reference has to be a virtual service reference. A virtual service reference can be used by an Enterprise Service Bus (ESB) to determine the endpoint of the service [55]. An endpoint is the concrete point, where a service can be reached.

The participant reference of the successful bidder is passed to the seller in the message link `completionNotificationLink` using the attribute `participantRefs`. Similar to a reference being passed using `bindSenderTo`, each passed reference is copied to the respective partner link. If a participant set is passed, the set is copied to the variable representing the set.

Realization of Participant Sets

A participant set is a set of participant references. The BPEL specification does not specify the XML type of a set of service references. Therefore, we define `sref:service-references` to be a `sequence` of `sref:service-reference` elements.

The `forEach` of BPEL may only iterate over numbers. Therefore, `forEach` activities iterating over participant sets are mapped to `forEach` activities iterating over a number and a nested `assign` activity copying the current service reference to a partner link.

Inclusion of Participant Grounding Details

In the participant behavior description, communication activities have no partner link and no operation assigned. With the participant grounding and the generation of partner links, partner links and operations are generated for each communication activity. Thus, the attributes `partnerLink` and `operation` are written upon mapping the activities.

More details on generating BPEL processes out of grounded BPEL4Chor choreographies can be found in [187].

4.3 Integration of BPMN and BPEL4Chor

There are two strategies for realizing a seamless integration between choreography modeling on the conceptual level and choreography modeling on the technical level. (1) The same choreography language is used on both levels. This demands that the language supports the requirements of both levels. (2) Different choreography languages are used on the different levels. In this case, transformations between choreographies must be possible, covering a bidirectional alignment. If changes are made to the model on one level, it must be possible to propagate these changes to the other level, often called round-trip-engineering.

We have discussed extended BPMN and BPEL4Chor as choreography languages following the interconnection modeling style. Given their support for a wide range of requirements, it is natural to consider these two languages in the discussion. Applying strategy (1) would mean that either BPMN is further extended with all the technical configuration possibilities or that BPEL4Chor is equipped with a graphical notation. Extending BPMN to an executable language has been pursued by Grosskopf [116] and Schreiter [195]. However, technical configurations as demanded by the requirements framework were largely neglected. On the other hand, BPEL4Chor could be equipped with a graphical notation. Tools such as NetBeans[1] or Active BPEL Designer[2] offer graphical editing of BPEL processes using proprietary notations. This is one of the reasons why BPMN was introduced. Strategy (2) would at least include transformations from BPMN to BPEL4Chor and from BPEL4Chor to BPMN. On top of that, change propagation would be needed.

In this section we are going to investigate the second strategy. Section 4.3.1 and 4.3.2 present the two directions of transformations and Section 4.3.3 discusses the results.

4.3.1 Mapping BPMN to BPEL4Chor

Transformations from BPMN to BPEL have been studied extensively in the literature. White [215] presented initial ideas on such a transformation without providing a concrete mapping definition. The BPMN specification [9] also outlines a BPMN-to-BPEL mapping. However, these approaches focus on high-level alignment and do not dive into semantic details. From an

[1] See `http://www.netbeans.org/`
[2] See `http://www.active-endpoints.com/active-bpel-designer.htm`

academic perspective, especially issues resulting from the incompatibility of block- and graph-structured languages were studied. A notable mapping has been published by Ouyang et al. [168]. Albeit restricted to a subset, the authors specify a formal mapping of BPMN models into BPEL processes and implemented it in a tool[3]. Their approach realizes discovery of certain process patterns, which, in turn, are mapped on structured activities in BPEL.

In [168] three classes of BPMN diagrams are distinguished: (i) those that can be translated using block-structured constructs only, (ii) those that require the use of control links and finally (iii) those that require event handlers, fault handlers and message passing within one process instance for realizing control flow dependencies. For instance, the occurrence of the workflow patterns arbitrary cycles and multi merge [17] make a diagram be of category (iii), as there is no direct support for these two workflow patterns in BPEL. We argue that the BPEL code resulting from (iii) is not usable as starting point for further refining it to process implementations. Therefore, we do not transform these kind of diagrams. Other than that, we use this algorithm as basis for extensions towards mapping extended BPMN to BPEL4Chor. This mapping involves additional challenges that will be tackled in this section.

Participant references and participant sets were introduced in Section 4.1 in order to distinguish different participants in a conversation. The direction of the associations connecting these specialized data objects with communication activities indicate the semantics. The following examples are going to show how these BPMN extensions relate to their counterparts in BPEL4Chor.

Figure 4.9 illustrates that an association from a participant reference data object leading to a sending task denotes that a message is sent to this participant. If an association leads to a receiving flow object (message event, invoke task), a message from this participant is expected. A participant set data object association with a multiple instance task or sub-process denotes that the loop will iterate over that participant set.

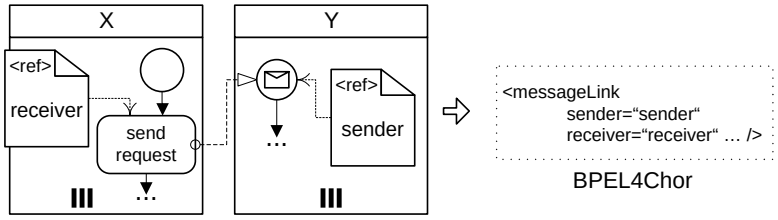


Figure 4.9: Sender and receiver

A directed association from a participant reference data object to a multiple instance sub-process denotes that the participant reference acts as loop counter.

Figure 4.10 shows an association from a receiving flow object to a participant set data object. This association denotes that a message is expected from an arbitrary participant and a reference to the sender of the message will be stored in the associated set. The actual participant reference in the set is represented by the participant reference data object associated with the flow object.

[3] See `http://www.bpm.fit.qut.edu.au/projects/babel/tools/`

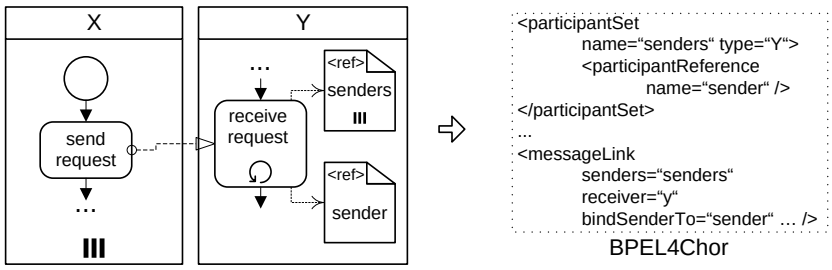


Figure 4.10: Storing the sender in a participant set

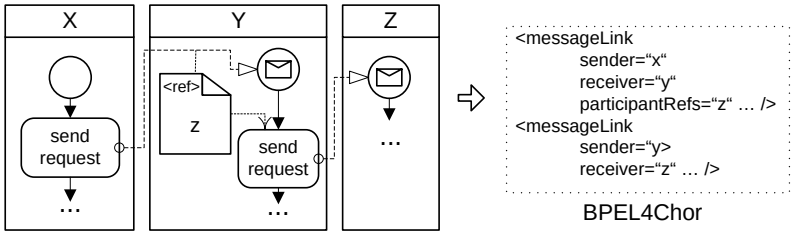


Figure 4.11: Passing a participant reference over the message flow

Besides communication activities, participant reference data objects and participant set data objects can be associated with message flows as presented in Figure 4.11. This realizes link passing mobility: The associated participant data objects are references passed over message flow.

Although the extended BPMN presented in Section 4.1 and BPEL4Chor have a large overlap in concepts covered, not all diagrams can be transformed to BPEL4Chor. The following BPMN elements cannot be mapped to BPEL4Chor: complex gateways, ad-hoc and transactional subprocesses, link, rule and multiple start events, all end events except the non-triggered ones, cancel, rule, link, multiple or non-triggered intermediate events and user, script, abstract, manual or reference activities.

General Approach. We largely base our transformation on the approach presented in [168] where a subset of BPMN is transformed to BPEL. This approach is based on the identification of *patterns* in the diagram that can be mapped onto BPEL blocks. One pattern is folded into a new activity, which is associated with the generated BPEL code. We extend these patterns with the elements used in the extended BPMN described above. Hence, we can use that transformation for transforming processes located in a pool, pool set or subprocess to their BPEL4Chor representation. Furthermore, we loosen certain restrictions as explained in the next subsection.

Multiple start and end events. In [168] it is assumed that there is only one start event and one end event in each process. We loosen this restriction and allow certain combinations of start events as well as multiple end events. If e.g. two start events are followed by a XOR-gateway, we fold this pattern to a BPEL pick element, where the attribute createInstance is set to “yes”. This scenario is captured by a generalized pick-pattern. Multiple end events are resolved by

merging the different branches into an inclusive gateway.

Inclusive gateways. We allow inclusive gateways if they occur in certain combinations with other elements and can be rewritten to AND- and XOR-gateways. In order to capture these combinations, the well-structured and quasi-structured patterns from [168] are extended. This means that our transformation can handle inclusive gateways in block-structured settings only.

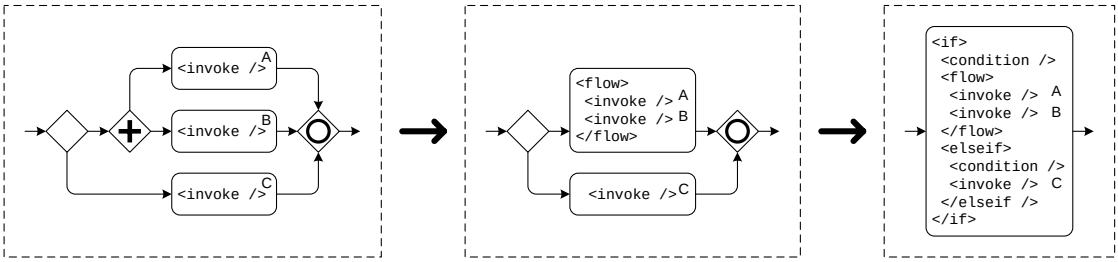


Figure 4.12: Dealing with inclusive gateways

Figure 4.12 illustrates an example. It exhibits two steps to transform a BPMN diagram involving multiple invoke activities to the corresponding BPEL representation. In the first step, an AND split gateway is translated to a BPEL flow, representing concurrent invocations of A and B. In the second step, the XOR split gateway is translated to an if construct in BPEL, so that either invocations of A and B are performed concurrently or C is invoked.

Other constructs. Each pool and pool set is mapped to a participant type. For a simple pool a participant reference with its corresponding type can be generated directly. Additional references are generated from participant reference data objects. The mapping of message flows to message links depends on the connected activities, the participant reference and participant data objects associated with these activities and the message data objects associated with the message flows. As the extended transformation removes elements from the model during the folding of the patterns, the topology has to be created beforehand.

The following pseudo code illustrates the overall approach.

1. Generate participant types in the topology from pools and pool sets
2. Generate participant references and participant sets from the participant reference and participant set data objects
3. Generate message links from the message flow, the associated participant reference and message data objects
4. Transform the processes within the pools and pool sets
   4.1. Generate the variables from the variable data objects
   4.2. Apply the extended transformation starting with the pattern for attached events

The approach has been implemented in the context of the Oryx project. All implementations are discussed in Chapter 6.

4.3.2 Mapping BPEL4Chor to BPMN

Despite the pressing demand for a mapping from BPEL to BPMN, it has not been tackled by recent research. It is assumed that such a *'transformation is relatively straight-forward'* [114] and *'that arbitrary BPEL processes can be mapped to BPMN using the flattening or the hierarchy-maximization strategy'* [185]. In fact, this assumption proves to be correct for numerous concepts. Essential BPEL activities can be mapped to BPMN directly and existing BPMN-to-BPEL mappings are bidirectional for many concepts. Nonetheless, there are a number of pitfalls in a BPEL-to-BPMN mapping. This section presents some of these issues.

There are a few activities that cannot be mapped at all, e.g. *termination handlers* and the *validate* activity. The concept of non-interruptive event handling, realized in BPEL through *event handlers*, it not present in BPMN, either. The majority of BPEL constructs can only be mapped partially.

Message-based Interactions. While the reception of messages can be realized in BPMN, a mapping of *reply* activities is not straight-forward. In particular, the question of a relation between message receiving and corresponding message sending constructs is not addressed in BPMN. The absence of a counterpart for BPEL *message exchanges* represents a pitfall in the mapping of message handling activities.

Exception Handling. In general, the exception handling framework of BPEL can be mapped to BPMN. Exception throwing activities correspond to error end events. A mapping to intermediate error events is no longer possible, as these events have been removed from the BPMN specification with version 1.1, due to their ambiguous semantics.

Another pitfall is the lack of a well-defined semantics for the propagation of exceptions in BPMN. In particular, it remains unclear, how exceptions in concurrent instances of an activity are treated. Please refer to [96] for a discussion of this issue.

Compensation Handling. BPMN's compensation activities are the counterparts for BPEL's *compensation handlers*. Moreover, the compensation order equals the one defined for BPEL. Nevertheless, semantics in case of invalid compensation triggers and the execution context of compensation activities is underspecified in BPMN.

In BPMN, compensation is triggered by compensation events. It might be reasonable to represent BPEL *throw* or *rethrow* activities by error events in BPMN. A similar approach is not feasible for compensation activities. In BPMN, control flow is directly passed to downstream activities after compensation has been triggered by a compensation intermediate event, while BPEL's *compensate* and *compensate scope* activities block the control flow until compensation has finished.

Dead Path Elimination. Dead path elimination (DPE) can be realized in BPEL using a flow activity, enabling parallel execution, and named control links (links, for short), which are applied to specify execution dependencies. Links always connect two activities and might assume

three different states: 'unset' (the initial state), 'true', or 'false'. In the case an activity with incoming links is enabled, the execution is delayed until all links are set to either 'true' or 'false'. Afterwards the join condition is evaluated and according to the result, the activity is executed or skipped. Subsequently, all outgoing links are set, either according to the transition condition (if the activity has been executed) or to 'false' (if the activity has been skipped). Links have to create an acyclic graph of dependencies and must not cross the boundary of repeatable constructs.

In order to realize DPE in BPMN the different states of a link have to be expressed. Two approaches seem to be reasonable. *Two* separate sequence flows represent both link states, 'true' and 'false' and a token is sent exclusively on one of these flows. Alternatively, *one* sequence flow represents the link. This approach would require non-local semantics to distinguish the states 'false' and 'unset' of the according link.

In any case, parallel gateways realize the parallel execution and the activation of sequence flows representing links in scenarios without transition conditions and join conditions. The activation of sequence flows representing links with transition conditions is modeled using inclusive gateways.

Moreover, a mapping of the join condition is not straight-forward. In order to activate outgoing sequence flows based on the activation of incoming sequence flows, we have to apply a complex gateway. All other gateway types require a clear separation of splitting and merging behavior. The BPMN specification allows for referencing the names of incoming branches in the incoming condition of the complex gateway. Although it remains unclear in the BPMN specification, we assume that this condition is evaluated, whenever a token arrives at the gateway. In addition, BPMN does not specify, whether tokens not satisfying the incoming condition are collected (they may satisfy the condition together with tokens arriving later) or discarded. A reasonable interpretation of semantics of the complex gateway would be to wait for synchronization of all incoming sequence flows and then activate the outgoing sequence flows according to the assigned conditions. Besides the fact that multiple outgoing sequence flows with different conditions cannot be modeled according to the BPMN specification, the notion of synchronization of the complex gateway is an open issue.

We summarize that BPEL processes involving *transition conditions* or *join conditions* cannot be mapped to BPMN, due to the ambiguous definitions of both, the inclusive gateway (merging semantics) and the complex gateway.

Process Instantiation. A classification of process instantiation mechanisms and an evaluation of BPEL and BPMN has been presented in [79]. In general, BPEL processes are always instantiated through a single message, received by a *receive* activity or by a *pick* activity. We refer to these activities as start activities, if they have been configured for instantiation purposes via the Boolean attribute *create instance*. With respect to multiple start activities, BPEL allows for a variety of different scenarios. According to [79], subscriptions are established either for all of the remaining start activities (multiple *receive* activities), or only for reachable start activities (after one *onMessage* branch has been triggered, subscriptions for the remaining *onMessage* branches are discarded). Further on, BPEL enables precise definition of expiration for these subscriptions. Timer-based (*event handlers*) or message-based (*pick* activity) triggers might remove

subscriptions, while they might also reside until message consumption or process termination.

In line with BPEL, BPMN processes are also instantiated by the reception of messages. In fact, simple BPEL processes can directly be mapped to BPMN, as *receive* activities and *pick* activities applied to instantiate the process have their counterparts in message start events and the event-based gateway (configured via its attribute *instantiate*).

However, process instantiation mechanisms of BPEL and BPMN are compatible only at a first glance. The analysis in [79] reveals fundamental differences with respect to the conjunction of multiple start events. In particular, there is no means to issue subscriptions for message events in the course of process instantiation. Therefore, there is only one way to model scenarios that require multiple start events to occur. That is, multiple start events are directly connected to an activity, which furthermore is configured via a dedicated attribute. In this case, it is waited for the occurrence of all events before the process instance is created. Please note that the actual configuration of the activity remains unclear, as there is no such attribute in the BPMN meta-model. The attribute *start quantity* is inapplicable in this context, as it does not relate to process instantiation.

Owing to the lack of a subscription mechanism, BPEL scenarios involving multiple start activities followed by different activities cannot be expressed in BPMN. Obviously, scenarios with complex dependencies between these subscriptions cannot be modeled in BPMN either. The Listing 13 shows an exemplary BPEL scenario that cannot be mapped to BPMN.

```
<sequence><flow>
  <sequence>
    <receive ... createInstance='yes'>
      <correlations> <correlation set='c' initiate='join' /> </correlations>
    </receive> ...
  </sequence>
  <sequence>
    <receive ... createInstance='yes'>
      <correlations> <correlation set='c' initiate='join' /> </correlations>
    </receive> ...
  </sequence> ...
</flow> ... </sequence>
```

Listing 13: Exemplary BPEL process instantiation scenario

Data Variables. BPEL defines variables, which belong to a certain scope (or the whole process, respectively). Further on, a scope enforces constraints with respect to data visibility — a variable is visible inside the scope containing its definition and all of its enclosed scopes. In addition, BPEL enforces lexical scoping of variables. In other words, variables of outer scopes can be hidden by introducing variables of the same name in an inner scope.

Data scoping is not addressed in BPMN, neither on the level of properties, nor for data objects. While the notion of input and output sets might be applied to clarify data visibility for activities, there is no way to restrict the set of data objects (or properties) accessed by expressions of sequence flows or gates. Therefore, especially the mapping of multiple BPEL variables with the same name is cumbersome. All constraints resulting from the lexical scoping of these

variables have to be implemented manually in BPMN at the expense of additional data objects.

4.3.3 Summary

In order to approach a feature-complete BPEL4Chor-to-BPMN mapping, BPMN semantics would need to be clarified (for instance for the complex gateway) and additional constructs would be necessary (for instance non-interruptive event-handling). It is foreseeable that these issues complicate BPEL4Chor-BPMN-round-tripping significantly. Despite the distinct goals of both languages, the vision of BPEL4Chor-BPMN-round-tripping requires all information to be mapped from BPEL4Chor to BPMN, even the low-level configuration. If any information is lost during transformation, it would have to be added again during the generation of BPEL4Chor choreographies out of the BPMN model. Nevertheless, it is reasonable to hide most of the configuration in attributes of the BPMN model, as it is done with information about message correlation.

The Business Process Modeler by eClarus[4] promises enabling of round-trip-engineering for BPMN and BPEL. The implemented mapping algorithm has been sketched by Gao [114]. It potentially involves restructuring of the BPMN process flow in order to create BPEL-isomorphic processes. However, this tool does not solve the abovementioned challenges in the two transformation directions.

We can conclude that full round-trip-engineering regarding BPMN and BPEL4Chor is not possible. By defining subsets of the languages (e.g. excluding compensation handling, event handling, complex gateways and other constructs), however, the two transformation directions could be realized.

4.4 Correlation Issues

When a participant engages in multiple conversations concurrently, it must be possible to relate an incoming message to other messages that were previously sent or received. It must be possible to relate such a message to the corresponding process instance. There are a number of mechanisms for implementing such correlation. For example, a correlation token such as a conversation identifier could be included in the message. Alternatively, and which is also most commonly used, a participant generates an identifier that has meaning to him, e.g. a purchase order id, and requires the other participants to include this identifier in the response messages as well. More advanced techniques are not based on mere equality of values but rather use value ranges, others use time-intervals or even moving time-windows to correlate incoming messages. A comprehensive catalog of correlation mechanisms can be found in [35].

This section focuses on the usage of identifiers for correlation only. It will be shown that using an extension to open nets, so called ν^*-nets, formal verification regarding proper correlation configurations can be carried out. ν^*-nets include the concepts of name creation and name passing as it is known from π-calculus [160]. The notion of *instance isolation* is defined, guaranteeing that two process instances of the same role will not participate in the same conversation.

[4] See `http://www.eclarus.com/products_soa.html`

4.4.1 Correlation Architecture

We will use a simple procurement example throughout this section: Buyers place orders at sellers who in turn respond with an order acknowledgement. As a third message exchange in the conversation, the buyer sends payment to the seller. This might be done e.g. through the transfer of credit card information.

While individual conversations are always bilateral in this example, we assume a number of buyers and sellers in the overall setting. All interaction is carried out through electronic message exchanges between the different organizations' information systems. Each information system has three ports for communicating with the outside world. There is one port for each message type: order, order acknowledgement and payment.

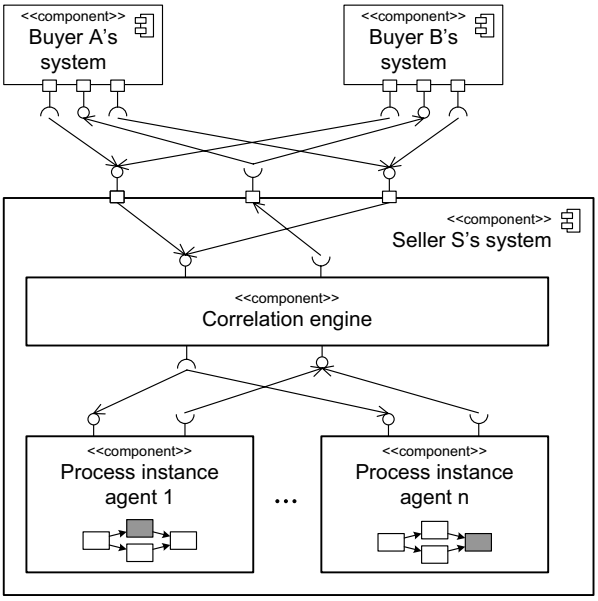


Figure 4.13: Architectural overview: Correlation engine and process instance agents

When looking into the internal structure of seller S's system, we see a set of process instance agents. The different process instances refer to the orders that are being processed concurrently. However, as the same ports must be used for different process instances, a correlation engine is necessary as additional component. We modeled that all communication between the process instance agents and the outside world must go through the correlation engine. We introduced message send and receive interfaces for the communication between the correlation engine and the process instance agents. Upon arrival of a message from outside S's system, this message is forwarded to the correlation engine, which in turn checks which process instance agent is subscribed to it. Depending on this, it forwards it to the corresponding process instance agent. Upon arrival of a new order, a new process instance agent is created. Figure 4.13 depicts these components and their dependencies using the UML notation [5].

Although correlation mechanisms are offered in state-of-the-art information systems, the correlation configuration of implementations is often not properly set. A possible outcome

could be that a message is routed to the wrong process instance agent or that subscription is already forbidden in the first place. BPEL defines different exceptions occurring in the context of erroneous correlation configuration. If correlation sets are not properly initiated before being used, a *correlationViolation* exception is thrown. Furthermore, two process instances must not subscribe to the same set of messages. Here, a *conflictingReceive* exception applies. Finally, if an incoming message can be matched for two process instances with different subscriptions, an *ambiguousReceive* exception is thrown. Especially the latter two exceptions only occur when dealing with concurrent conversations.

4.4.2 Formal Model

In open nets (cf. Section 2.4), two tokens located at the same place cannot be distinguished from each other. Therefore, two messages represented by tokens residing on the same place cannot be distinguished from each other.

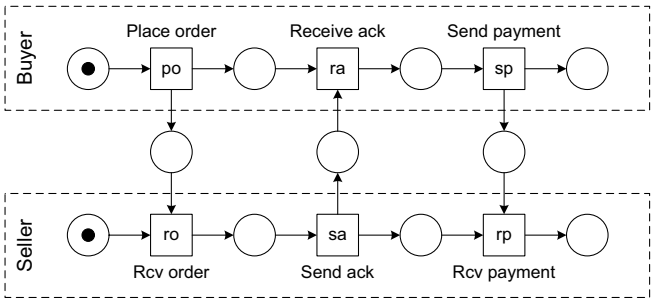


Figure 4.14: Composition of two open nets

Figure 4.14 describes the sample choreography from section 4.4.1 as composition of open nets representing the seller and the buyer. The three ports for orders, order acknowledgements and payments are modeled as places. There is only one firing sequence allowed: 'Place order', 'Rcv order', 'Send ack', 'Rcv ack', 'Send payment', 'Rcv payment'.

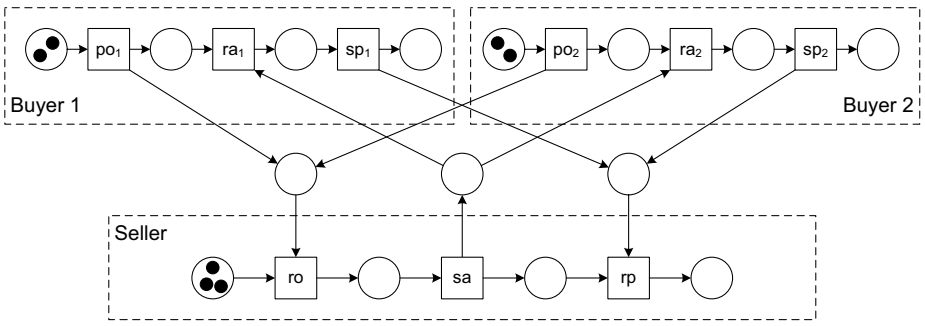


Figure 4.15: Concurrent conversations

Figure 4.15 shows a variation of the first composition. Now, we introduced a second buyer and more tokens. This represents concurrent conversations between the buyers and the seller.

However, we cannot distinguish which buyer will get which acknowledgement. Such correlation information is not included in this representation.

ν^*-Nets

We introduce ν^*-nets for representing correlation information. They are an extension for open nets with name creation and name passing. Name creation and passing has been extensively studied in process algebras such as π-calculus. However, there is a subtle difference between the usage of names in π-calculus and ν^*-nets: π-calculus works with name substitution. In contrast to this, we distinguish names and *variables* in ν^*-nets. Tokens flowing through the net carry names, while the unmarked net is labeled with variables. Upon firing, to each variable a name is assigned. We denote the (infinite) set of names with Id and the set of variables with Var. Name creation leads to the introduction of fresh names into the marking. A special variable ν is reserved for this purpose ($\nu \in Var$).

The flow connections between places and transitions are labeled with vectors of variables that can have zero or more components. We introduce Var^* as the set of all variable vectors. We are going to use the symbols $\in$ and $||$ for denoting the containment of a value in a vector and the length of a vector. E.g. $b \in (a, b, c)$ and $|(a, b, c)| = 3$.

Definition 4.1 (ν^*-net) A *ν^*-net* N is a tuple $N = (P, P_C, T, F, m_0)$ where

- P and T are disjoint sets of places and transitions,
- $P_C \subseteq P$ is the set of communication places,
- $F : (P \times T) \cup (T \times P) \rightarrow Var^*$ is a partial function where to each flow connection between places and transitions a variable vector is assigned and
- $m_0 : P \rightarrow \mathcal{MS}(Id^*)$ is the initial marking where to each place a multi sets of name vectors is assigned.

□

We introduce the auxiliary functions $pre, post : T \rightarrow Var$ for denoting input and output variables of a transition, where $pre(t) := \{v \in Var \mid \exists p \in P\ (v \in F(p, t))\}$ and $post(t) := \{v \in Var \mid \exists p \in P\ (v \in F(t, p))\}$. We also introduce $var(F)$ as the set of all variables in the net, i.e. $var(F) := \{v \in Var \mid \exists (o_1, o_2) \in dom(F)\ (v \in F(o_1, o_2))\}$. We assume all ν^*-nets to satisfy the following conditions:

- For every place p all variable vectors assigned to flow connections originating in p or targeting p have the same length, i.e. $\forall p \in P\ [\exists n\ (\forall (o_1, o_2) \in dom(F)\ [p \in \{o_1, o_2\} \Rightarrow |F(o_1, o_2)| = n])]$.
- ν does not occur in variable vectors assigned to flow connections targeting transitions, i.e. $\forall t \in T\ [\nu \notin pre(t)]$.
- All variables occurring in variable vectors assigned to flow connections originating in transitions occur in at least one of the variable vectors assigned to a flow connection targeting that transition, i.e. $\forall t \in T\ [post(t) \setminus \{\nu\} \subseteq pre(t)]$.

- No variable may be contained twice in the same variable vector.

Tokens flowing through ν^*-nets are colored. They represent vectors of names, where Id^* is the set of (potentially empty) name vectors. We denote the set of names in a marking m as $S(m)$, where $S(m) := \bigcup_{p \in P} \{id \in ids \mid ids \in m(p)\}$.

We assume that a ν^*-net satisfies the following condition: Each name vector contained in a marking has the same number of components as the variable vectors assigned to incoming and outgoing flow connections to that place.

We introduce *modes* as assignment of names to input and output variables of a transition, $\sigma : (pre(t) \cup post(t)) \rightarrow Id$. For every mode σ we introduce a corresponding vector mode $\sigma^* : Var^* \rightarrow Id^*$ assigning name vectors to variable vectors, where $\sigma^*((v_1, \ldots, v_n)) = (\sigma(v_1), \ldots, \sigma(v_n))$ and $\sigma^*(()) = ()$.

Definition 4.2 (Enablement and Firing) Let (P, P_C, T, F, m_0) be a ν^*-net and m a marking. A transition $t \in T$ is enabled with mode σ if $\sigma(\nu) \notin S(m)$ and $\forall p \in P\ [(p,t) \in dom(F) \Rightarrow \sigma^*(F(p,t)) \in m(p)]$. The reached marking after firing of t is m', where $m'(p) := m(p) - \{\sigma^*(F(p,t))\} + \{\sigma^*(F(t,p))\}$. We denote this as $(P, P_C, T, F, m) \overset{t(\sigma)}{\rightarrow} (P, P_C, T, F, m')$ or $m \overset{t}{\rightarrow} m'$ for short. □

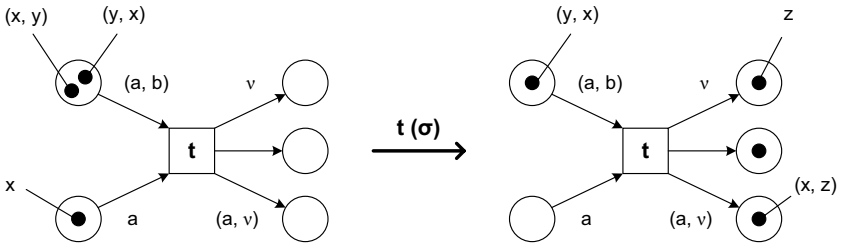


Figure 4.16: Sample ν^*-net and firing of t

Figure 4.16 shows a ν^*-net on the left. Here, transition t is enabled with mode σ, where $\sigma(a) = x$, $\sigma(b) = y$ and $\sigma(\nu) = z$. Firing of t with mode σ leads to the marking depicted on the right. As Figure 4.16 shows, we omit inscriptions of flow connections and token descriptions in the diagram whenever the corresponding variable and name vectors have zero components.

In order to describe choreographies with ν^*-nets, we can reuse open composition of open nets in the context of ν^*-nets. Furthermore, open composition of two nets can canonically be extended to composition of a set of nets.

Figure 4.17 depicts the open composition of ν^*-nets for the example of section 4.4.1. Two correlation identifiers were used in the model. A fresh name is created upon firing of transition po. Note that the two tokens produced on the output places of po will carry the same name. In the following, variable o is used to pass this name through the net. Although such a consistent naming of variables in a ν^*-net is not mandatory, it increases the readability of the model. The most important use of variables o is at transition ra: Here, both input tokens have to carry the same name. I.e. synchronization only takes place if the message sent by the seller is awaited for by the buyer.

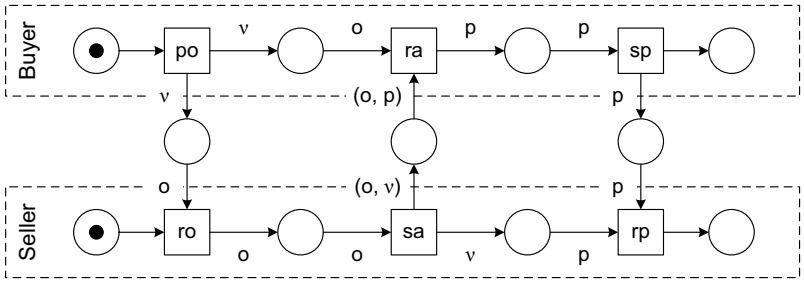


Figure 4.17: Open composition of ν^*-nets

Another interesting situation already occurs at transition ro. While the token produced by po carries a name, the second token consumed does not carry any name. The seller accepts any message from the buyer and passes on the received name for later use.

A second fresh name is created upon firing of transition sa. For properly correlating the different messages in the conversation the creation of one fresh name would have been enough. However, using different correlation identifiers is quite common in real-world settings. While the first part of the conversation makes use of an order identifier o, a switch towards using a payment identifier p is made during the conversation. In addition to different phases as reason for different correlation identifiers, the different participants typically want to define their own correlation identifier.

4.4.3 Instance Isolation

This section is going to introduce the notion of instance isolation for compositions of ν^*-nets. These compositions typically represent (all possible paths of) exactly one conversation. As a participant of a particular role might be involved in several conversations at the same time, it must be checked whether the conversations are isolated from each other. We want to ensure that two process instances for the same ν^*-net cannot participate in the same conversation. In the remainder of this section we will present how this can be checked. A process instance must not compete for the same message with another process instance. By competition (or conflict) we mean that two different process instances are able to consume exactly the same message. This can be checked by analyzing two concurrent conversations.

A process instance joins a conversation as *initiator* or *follower*. As initiator it sends a message prior to receiving a message. As follower it is the other way round. In the case of a follower, it often occurs that the message received can be consumed by any process instance. I.e. in this particular case, it should be allowed that different process instances compete for the same message. However, once a message was sent or received, there should be no competition for messages with other process instances any longer. This must be considered in the instance isolation analysis.

Figure 4.17 from the previous section illustrates the distinction between initiators and followers. Here, the seller acts as follower and the incoming purchase order might be processed by any process instance. More details on initiators and followers can be found in [35], where a set

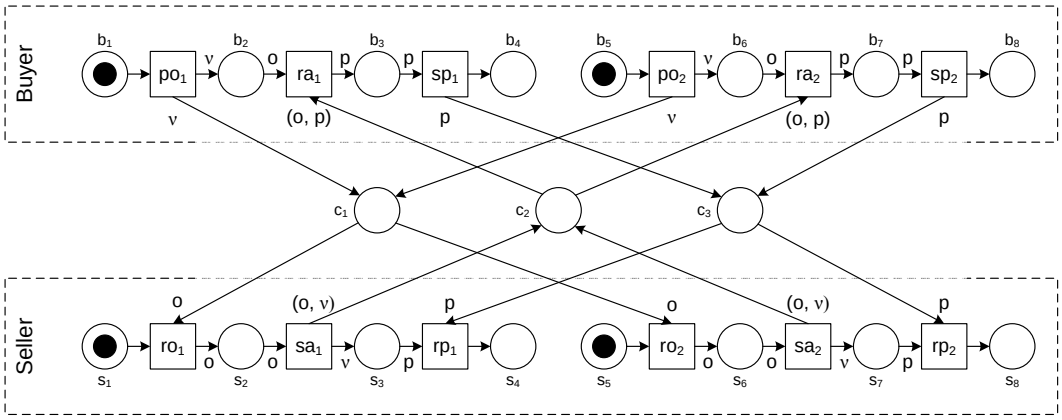


Figure 4.18: Representing two concurrent conversations

of common correlation scenarios are listed.

For instance isolation analysis, we are going to represent two concurrent conversations by duplicating the ν^*-nets. Figure 4.18 illustrates this for our example. At the heart of the analysis, we need to identify those situations where two receive transitions belonging to the same role are enabled for the same message in different conversations.

The basic idea is to use reachability of markings for detecting these situations. We need to find those markings where two competing receive transitions are enabled for the same token on a shared communication place.

Definition 4.3 (Competition marking) Let (P, P_C, T, F, m_0) be the open composition of the ν^*-nets $N_1, \ldots, N_n$. A marking m is a *competition marking* regarding N_i and N_j iff $i \neq j$ and there exist transitions $t_1 \in T_i, t_2 \in T_j$ that share a communication place as input place, i.e. $P'_C = {\bullet}t_1 \cap {\bullet}t_2 \neq \emptyset$, and that are enabled in m with modes σ_1 and σ_2 and $\sigma_1^*(F(p, t_1)) = \sigma_2^*(F(p, t_2))$ for some $p \in P'_C$. □

Figure 4.19 shows the marking $[b_2, b_5, c_1, s_1, s_5]$. ro_1 and ro_2 share the communication place c_1, i.e. $P'_C = \{c_1\}$. The token residing on place c_1 could be consumed by either transition ro_1 or transition ro_2, due to $\sigma_1^*(F(c_1, ro_1)) = \sigma_2^*(F(c_1, ro_2)) = (x)$. We can conclude that the marking $[b_2, b_5, c_1, s_1, s_5]$ is a competition marking.

In this marking, however, competition should actually be allowed as the two sellers have not entered the conversations yet. The assignment whether buyer 1 talks to seller 1 or seller 2 has not been decided yet. Therefore, we must distinguish between situations where competition is allowed and situations where a message was already sent or received by the respective process instance. This can be detected by checking whether there is a transition sequence that leads to the marking and contains a corresponding communication transition. If this is the case, we say that the respective process instance is *conversation-engaged* in that marking.

Definition 4.4 (Conversation-engagement) Let (P, P_C, T, F, m_0) be the open composition of ν^*-nets $N_1, \ldots, N_n$. N_i is *conversation-engaged in marking* m iff there exists a transition

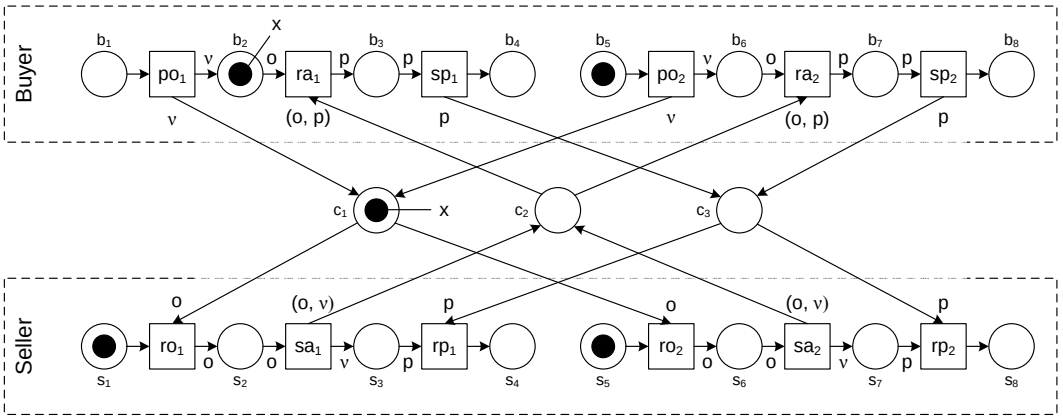


Figure 4.19: Competition marking

sequence from m_0 to m containing a transition $t \in T_i$ for that holds $(\bullet t \cup t\bullet) \cap P_{C1} \neq \emptyset$. □

As already mentioned above, the strategy of instance isolation is to duplicate the ν^*-nets and then check for competing receive transitions. Duplications of ν^*-nets are straightforward: All transitions, all places (except communication places), all flow relationship and the initial marking are copied.

Definition 4.5 (k-composition) Let $N_1, \ldots, N_n$ be ν^*-nets and $D_{i1}, \ldots, D_{ik}$ the k duplications of N_i. Then the *k-composition* is the open composition of $D_{11}, \ldots, D_{nk}$. □

Finally, we are able to define instance isolation based on k-composition, competition markings and conversation-engagement.

Definition 4.6 (Isolation regarding k instances) Let $N_1, \ldots, N_n$ be ν^*-nets. *Isolation regarding k instances* is given for the k-composition iff there is no competition marking m regarding D_{ia} and D_{ib} in which D_{ia} or D_{ib} are conversation-engaged. □

Definition 4.7 (Complete Instance Isolation) Let $N_1, \ldots, N_n$ be ν^*-nets. *Complete instance isolation* is given iff for every $k \geq 2$ isolation regarding k instances is given. □

While it is possible to check isolation regarding k instances for a given k, we are not able to check complete instance isolation directly. We will show that checking isolation regarding 2 instances has implications for k instances under certain conditions.

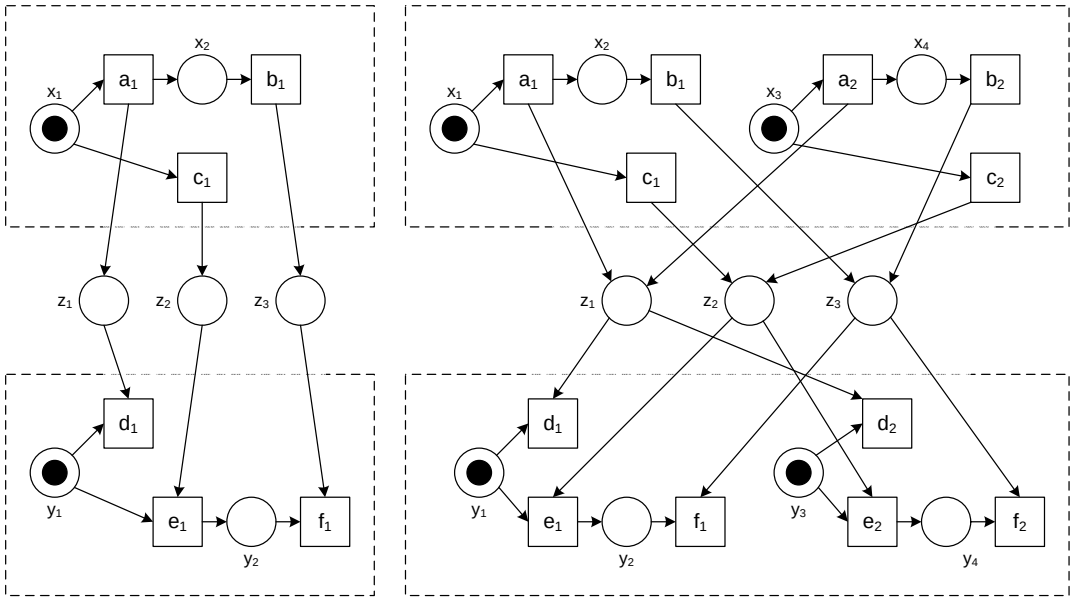
Theorem 4.1 *Isolation regarding 2 instances in the presence of reachability of all transitions in the 1-composition implies complete instance isolation.*

This theorem can be proven by assuming a choreography where all transitions in the 1-composition are reachable, isolation regarding 2 instances is given and isolation regarding k instances (for some $k > 2$) is not given. We will show that this leads to a contradiction.

As isolation regarding k instances is not given, we know that there is a competition marking m with "problematic" messages on one (or several) communication places. This implies that either these message do not carry any distinguishing identifier or this identifier is ignored by the

process instances. We know that at least one of the process instances is conversation-engaged in m and receive transition t belonging to this process instance is enabled.

All transitions in the 1-composition are reachable, including transition t. Therefore, sufficient messages are produced onto the communication places for t to fire in the 1-composition. Furthermore, it must be possible that the process instance is conversation-engaged when t is enabled. We can conclude that if it is possible to enable transition t in the 1-composition, it is also possible to reach a marking m' in the 2-composition where both t and the copied transition t' of the second process instance are enabled for the same messages. m' must be a competition marking then and therefore isolation regarding 2 instances is not given, q.e.d.



(a) 1-composition (b) 2-composition

Figure 4.20: One process instance interacts with two other instances in the 2-composition

Figure 4.20 shows a 1-composition and the corresponding 2-composition. In this example, isolation regarding 2 instances is given although no fresh name is used. The lower left process instance can first receive a message from transition c_1 and then a message from transition b_2. That way, it interacts with two process instances. Furthermore, no marking is reachable where f_1 and f_2 are both enabled. When adding a third process instance, it becomes obvious that isolation regarding 3 instances (and higher) is not guaranteed any longer. The third process instance could produce a second message on z_2, leading to a competition marking.

4.4.4 Discussion

Missing correlation configuration is the typical reason for non-isolation of process instances. However, we need to distinguish between synchronous and asynchronous communication. Often, request/response interactions are carried out synchronously. In this case, there are corre-

lation mechanisms in place in a lower level of the protocol stack. E.g. a TCP connection is exclusively used for exchanging request and response. Synchronous invocations can be properly modeled by creating a fresh name and passing it along with the two messages. A discussion about how name creation and passing can be used to formally model such scenarios was already reported in [85], where π-calculus was employed.

As an alternative to using one channel and including correlation information as part of the message contents (as it is the case in WS-style correlation), a different approach is advocated by the architectural style Representational State Transfer (REST [112]). As discussed in [222], a dedicated channel instance (in the form of a URI) is created for an expected response message. Such a URI corresponds to the names used in ν^*-nets and in π-calculus. Therefore, ν^*-nets and π-calculus can be used to reflect both a REST-style correlation mechanism as well as a WS-style mechanism. However, one might argue that π-calculus more closely resembles the REST-style while ν^*-nets more closely resemble the WS-style due to the presence of communication places.

One might also argue that a human modeler can easily detect missing or erroneous correlation configuration in choreographies and that hence automatic checking of this property is not needed. However, it must be considered that choreographies representing real-world scenarios can easily contain hundreds of communication places, where manual model verification becomes error-prone.

The definition of instance isolation presented in this section assumes that every role, represented by a ν^*-net, is executed at most once in a conversation. However, there might be scenarios where the same role is instantiated multiple times. Imagine a procurement scenario where a shipper has to be selected among a set of shippers. As part of that, prices and shipping conditions need to be requested from all shippers. In this scenario there would be one role representing all shippers and the technique presented in this section would only check isolation of a set of process instances involved in one conversation regarding a set involved in another conversation.

In terms of computational complexity, the proposed technique suffers the usual drawbacks of reachability analysis in the presence of concurrency. The duplication of conversations in our technique worsens this situation as it significantly increases the number of reachable markings.

Correlation is a well-known concept especially in enterprise systems, where long-running conversations are a frequent phenomenon. Hohpe and Woolf documented a set of architectural patterns for such systems in [127] also including message correlation. Several web services standards introduce correlation identifiers as first-class citizens. BPEL [108] includes *correlation sets* and the Web Services Choreography Description Language (WS-CDL [136]) includes *identity tokens*. WS-Addressing [118] is an extension to the SOAP messaging format introducing a *replyTo* and *faultTo* field into the message header. An overview over recurrent correlation use cases can be found in [35]. The same paper uses these correlation patterns to assess BPEL and WS-CDL.

De Pauw et al. present a technique for mining conversations from message logs in [171]. As part of that, correlation identifiers are identified using heuristics.

The formal model used in this section extends ν-nets as presented in [208]. Here, each token carries one name. Name matching is applied upon synchronization, i.e. in case a transition has more than one input place. ν-nets are actually a subclass of ν^*-nets where each name vector and variable vector is of length 1.

ν^*-nets have the interesting property that reachability implications based on corresponding open nets are possible. As exactly one token is consumed from each input place and exactly one token is produced on each output place, ν^*-nets are merely more restrictive than open nets regarding enablement of transitions. Therefore, we can conclude that if a certain transition or marking in the corresponding open net is not reachable, then the corresponding transition or marking is not reachable in the ν^*-net, either. Furthermore, if the open net is bounded, then the ν^*-net will also be bounded.

The observation that a criterion might hold for a case $k = 2$ but does not hold for some other k, has extensively been studied in the case of k-*soundness* [206, 207], a correctness criterion for workflow nets that is a natural extension to the classical soundness (or 1-soundness) notion [18]. In this context, generalized soundness means that k-soundness is given for any k. k-soundness is different to instance isolation as the same net structure is shared by all instances and therefore the influence of the instances on each other is much higher. In the case of instance isolation, only a small portion of the net structure is jointly used by all instances, namely the communication places. Furthermore, the notion of conversation-engagement further adds to the decoupling of the instances. Instance isolation checking cannot be mapped to k-soundness checking as the underlying formal model for instance isolation heavily relies on name creation and name passing capabilities which are not present in workflow nets.

4.5 Discussion and Summary

In addition to the introduction of the notion of instance isolation, this chapter has shown that slim extensions to existing languages are sufficient to broaden their support for the requirements from chapter 3. While these extensions seem promising at a first glance, practical insights into the usage of these languages have already revealed major drawbacks when it comes to modeling complex choreographies. We will use extended BPMN as basis for further discussions, although similar issues also apply to BPEL4Chor.

4.5.1 Redundancy

Choreography modeling on the conceptual level centers around models that evolve and become more complex over time. A good modeling language allows the modeler to reflect decisions in the model without much effort. The number of changes needed should be minimal.

The nature of interconnection modeling requires all control-flow- and data-flow-related decisions to be reflected on a per-role basis. If a modeler wants to define that the auctioning phase finishes before payment and delivery, she has to reflect this decision for the seller role and the buyer role, as both roles are involved in all complex interactions. Also the decision that payment should be realized through two messages, corresponding communication activities and the control flow must again be added to both roles. It becomes even worse when introducing more complex branching structures and loops into the model. Here, modelers need to carefully think about how it affects the different roles.

A better suited choreography language avoids redundancy. It would only require to reflect each decision once. Redundancy, in contrast, leads to inconsistencies within one model, which

in turn often leads to modeling errors.

4.5.2 Over-Specification

On the way to a refined choreography model, the modeling language imposes certain syntactic restrictions on which models are to be considered valid and which not. These restrictions therefore reflect the necessary modeling decisions that have to be made. An example for an essential modeling decision is the ownership of choices. If several alternative options exist, the modeler needs to at least specify who is responsible for that decision or if there can be a shared understanding among the different participants which alternative to take.

On the other hand, those restrictions that require an unimportant modeling decision need to be avoided. For instance, BPMN supports to specify the conditions for creating a new process instance. So called start events can have a defined trigger. This forces the modeler to decide whether a new process instance should be created in the course of a conversation or whether previously created process instances might become involved in a conversation. BPMN even goes one step further and allows the definition of multiple process models per role involved in the same choreography. For instance, each bidding message exchange initiated by a buyer could be done in a separate process instance.

We argue that the decision about process instance boundaries and process instantiation should be left open to each individual participant. Boundaries of process instances are not to be decided on in a choreography. Choreographies should rather only focus on allowed conversations.

4.5.3 Choreography Modeling Anti-patterns

In addition to the issues identified in the previous section, there are a number of anti-patterns that can be observed in a large number of interconnection models, often leading to modeling errors. Anti-patterns are parts of interconnection models that should be avoided during choreography modeling. The list of anti-patterns presented in this section was collected by observing students modeling BPMN. However, they are expected to be common also for other modeling languages following the interconnected modeling style.

We can distinguish three categories of anti-patterns: those related to decision-making (*D*), to ordering constraints (*O*) and to process instance creation and termination (*I*).

D1. Incompatible branching behavior. Two types of choices can be distinguished: Data-based choices (or *exclusive choice* [17]) and choices driven by the environment (*deferred choice* [17]). The fundamental difference between them is the ownership of the choice. While a data-based choice is decided *within* a process instance based on the process data, an environment-driven choice is based on external triggers.

Imagine two interconnected roles as illustrated in Figure 4.21. As both observable behavior models contain a data-based choice, it can be decided for each participant individually which branch to take. If the bidder decides to take the upper branch (credit card payment) and the seller takes the lower branch (payment by bank transfer), the conversation will deadlock as the seller will wait infinitely for the bank transfer payment message to arrive.

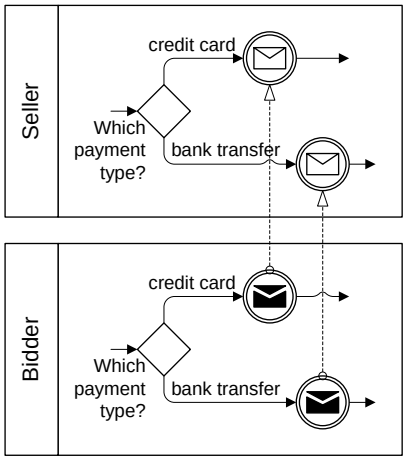


Figure 4.21: D1. Incompatible branching behavior

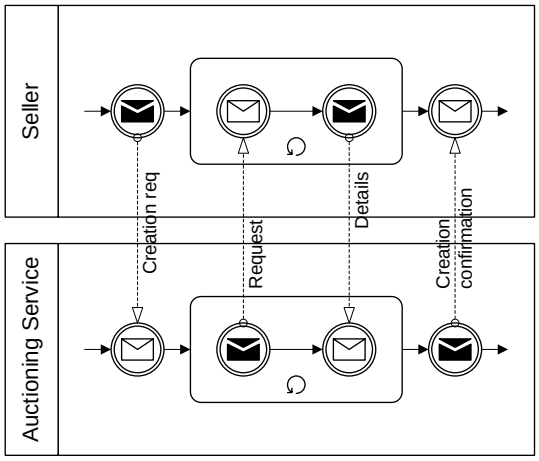


Figure 4.22: D1. Incompatible branching behavior (loops)

BPMN offers a popular shortcut notation for representing loops. Instead of using XOR gateways, subprocesses with a loop marker can be used. Figure 4.22 shows an example where the seller first sends an auction creation request and then the auctioning service keeps on requesting further details before finally sending a creation confirmation. In this example, the auctioning service controls the loop in the sense that it decides whether to execute the communication activities within the loop subprocess again. In contrast to this, the seller depends on the auctioning service's choice and has to react depending on whether another detail request comes in or a creation confirmation comes is.

The corrected models for Figure 4.21 and Figure 4.22 can be found in Figure 4.23 and Figure 4.24. In both cases, the event-based XOR gateway was used to properly reflect that the decision is made by the environment. Especially the correct representation of loops where the environment decides whether to stay in the loop or leave it seems to be difficult for human modelers. Based on the modeling decision whether a choice is data-based or event-based the resulting models look very different. This is counter-intuitive for many modelers.

Assuming individual decisions, an interesting situation arises in the presence of non- observable decisions, as in Figure 4.21. As the decision made by the seller cannot be observed by the bidder, the bidder cannot know which message is expected. Non-observable choices often lead to uncontrollable models, where no models exist that are compatible with it (cf. Section 2.5.2).

This only holds true if the choices are actually independent from each other. If there is a shared understanding of which branch to take, this problem does not arise. E.g. if credit card payment has to be chosen for all items above a certain price or if the payment type was already agreed upon earlier in the conversation, both participants come to the same decision.

D2. Impossible data-based decisions. In many cases, data-based decisions are used with conditions that actually cannot be evaluated by the respective role. The data the decision is based on might simply not be available to that role. Figure 4.25 depicts an example, where a

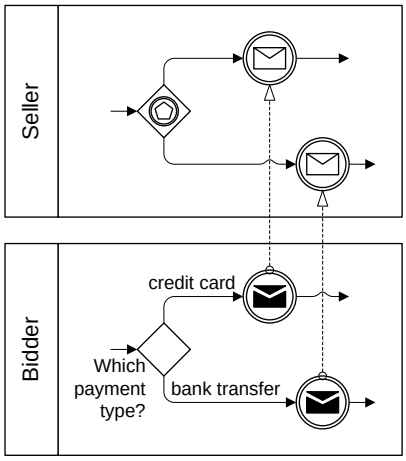


Figure 4.23: Corrected model for Figure 4.21

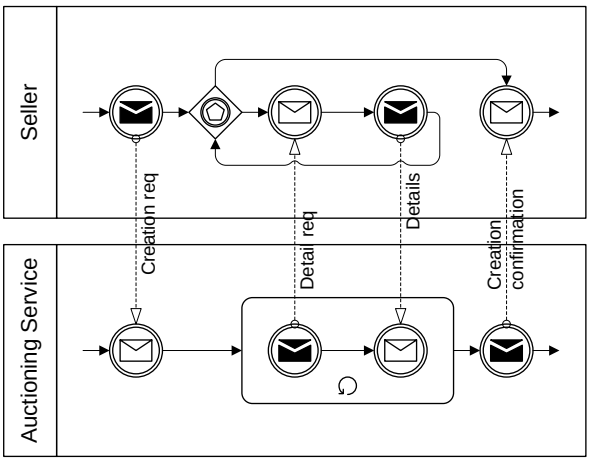


Figure 4.24: Corrected model for Figure 4.22

bidder decides whether she has won the auction. We assume that the bidder cannot know who won the auction and needs to wait for the notification by the auctioning service.

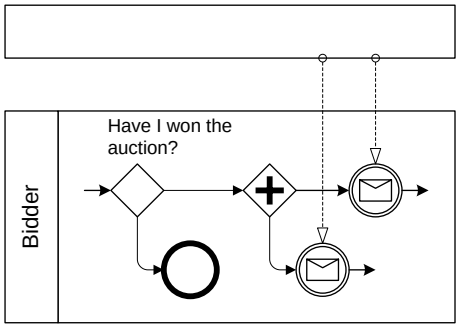


Figure 4.25: D2. Impossible data-based decisions

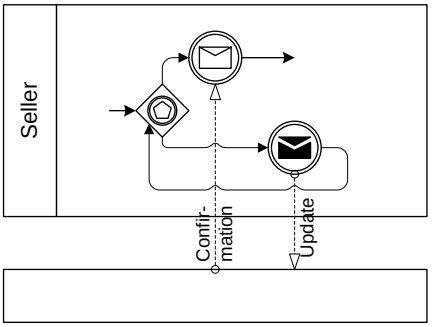


Figure 4.26: D3. Mixed choices (with syntax error)

D3. Mixed choices. Mixed choices appear in those scenarios in which there is a choice between a message send activity and a message receive activity in such a way that the receive alternative can only be chosen if an incoming message is actually present. Figure 4.26 shows an example where a modeler tried to express that a seller can alter the configuration of an auction as long as it has not been started by the auctioning service (which is indicated by a notification message).

The BPMN diagram in Figure 4.26 is syntactically not correct. An event-based gateway must only be followed by catching intermediate events or message receive tasks. That way, BPMN does not support modeling mixed choices. In contrast to this, many cancellation or update interactions actually require something at least close to mixed choices. Modeling this in

BPMN correctly is very challenging and even experienced modelers have difficulties doing so.

While mixed choices cannot be represented directly in BPMN and other interconnection modeling languages, different workarounds are possible, e.g. involving intermediate timer events. The main challenge is that both participants in a mixed choice can send messages at the same time. There are a number of strategies for resolving such situations. Section 5.4.5 will discuss a number of these strategies in more detail.

Now we look at two anti-patterns regarding ordering of interactions.

O1. Contradicting sequence flow. Interconnection modeling requires redundant modeling of ordering constraints between interactions. That way, modelers come up with models where sequence flow relationships defined for different roles contradict each other. Figure 4.27 illustrates an example where the seller waits for the payment before sending out the goods and the bidder waits for the goods before issuing the payment. This is a classical deadlock situation where both participants would wait infinitely for the respective message to arrive.

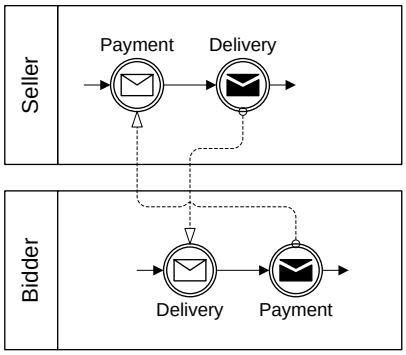


Figure 4.27: O1. Contradicting sequence flow

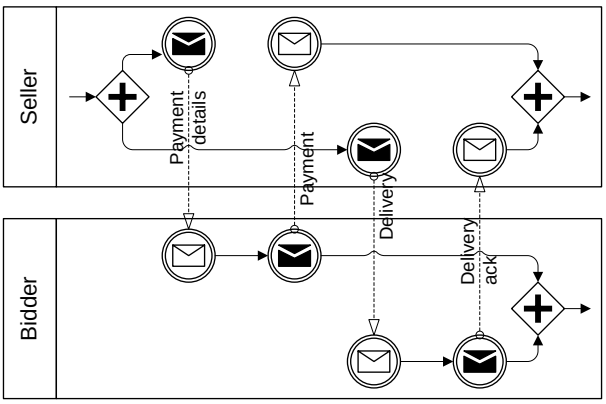


Figure 4.28: O2. Incomplete sequence flow

O2. Incomplete sequence flow. Some modelers simply omit sequence flow dependencies as illustrated in Figure 4.28. The rationale behind this is that the dependencies between the communication activities are implicitly given through their counterparts within the opposite role. While this assumption is partly valid, the resulting models are often wrong (as it is the case for Figure 4.28). Due to the missing sequence flows the diagram now contains two separate process models for the seller. The two receive activities serve as entry points into the second model. The model would therefore be expanded with two start events and the semantics for this is that the execution of any of these two start events triggers a new process instance. Therefore, the second process model would deadlock.

O3. Uni-lateral sequentialization. Some modelers come up with models, where the ordering constraints for the different roles are not equally permissive. Figure 4.29 shows that the seller

can handle payment and delivery in parallel while the bidder first completes the payment before awaiting the goods. The observable behavior model for the seller is more permissive than that of the bidder as it allows to handle the delivery before the payment has been completed.

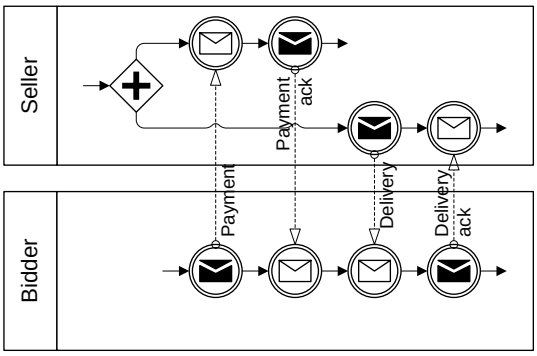


Figure 4.29: O3. Uni-lateral sequentialization

This anti-pattern does not cause a deadlock. However, the modeling decision of first handling the payment is only reflected in the seller's model.

Two anti-patterns regarding process instance creation and termination can be identified.

I1. Optional participation. Languages such as BPMN allow the specification of when to create a process instance. In the context of choreographies, incoming messages are a typical trigger for process instantiation. In contrast to this, a plain start event represents that it is left unspecified when a process instance is created and that instantiation will eventually happen "magically". In the presence of optional participation of a role, as illustrated in Figure 4.30, this becomes problematic. Here, the financial institution will deadlock if it is not involved in the conversation.

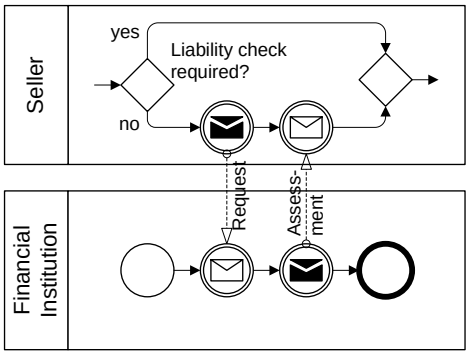


Figure 4.30: I1. Optional participation

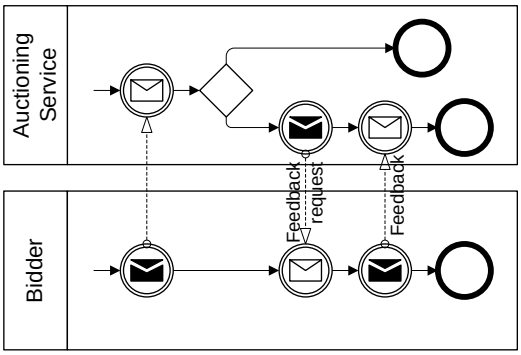


Figure 4.31: I2. Not-guaranteed termination

I2. Not-guaranteed termination. A typical anti-pattern related to the termination of process instances can also be seen in the context of optional participation in a certain part of a choreography. Figure 4.31 illustrates an example, where a bidder deadlocks if the auctioning service does not send a feedback request. The bidder cannot know whether the conversation has already ended or not.

Further anti-patterns have been described by van der Aalst et al. in [13], covering some of the anti-patterns that have been presented in this section.

4.5.4 Conclusion

The choreography anti-patterns from the previous section have shown that obviously it is not easy to create correct choreographies using interconnection modeling languages such as BPMN. It is an interesting question what the reason for these difficulties are. Different explanations could be put forward. (1) The BPMN users are not trained well enough. Otherwise they would be aware of the pitfalls in choreography modeling and would have avoided them. (2) The BPMN tools do not support the modelers well enough. They would have suggested the correct representation. (3) There is something wrong with the modeling language. A good language would simply not have these pitfalls and directly support modelers in the creation of correct models.

(1) The anti-patterns were distilled from models created by undergraduate and postgraduate students that previously had intensive training in BPMN and other modeling languages. They knew the workflow patterns such as explicit choice and deferred choice in detail and had extensive experience with BPMN's execution semantics. Still they created erroneous models.

(2) While most students use simple drawing tools such as Microsoft Visio[5] that do not provide any BPMN-specific support, others use more sophisticated BPMN tools such as Oryx[6] that include step-through-simulation and deadlock detection mechanisms. Obviously, these additional functions were not sufficient for getting around the pitfalls.

(3) It seems obvious that alternatives to interconnection modeling must be investigated further. Such an alternative would need to avoid redundancy regarding ordering constraints, provide easier means to specify proper branching structures and deal with instantiation and termination elegantly.

In chapter 3 we have seen that the interaction modeling style is an alternative to the interconnection modeling style. The main difference is that interactions are seen as atomic building blocks that are related using global dependencies. The following chapter will investigate how interaction modeling helps to avoid the anti-patterns presented in this section.

[5] See `http://office.microsoft.com/visio/`
[6] See `http://oryx-project.org`

Chapter 5

Interaction Models

UML Communication Diagrams, BPSS and WS-CDL follow the interaction modeling style (cf. Section 4.5). UML Communication Diagrams use textual annotations for expressing the behavioral dependencies between the interactions. Through notational convention, sequences, alternatives, parallelism and iterations can be represented, always leading to block structures. BPSS follows a graph-based approach allowing alternative branches as well as concurrency in choreographies. WS-CDL comes with a set of control flow constructs also covering sequences, alternatives, parallelism and iterations. Section 4.5 has revealed the limitations of these three languages. All lack the support for decomposition of interactions, reusability of choreographies and proper reflection of the ownership of choices.

Modeling choreographies on the conceptual level deserves special attention. This is due to the fact that the pitfalls discussed in the previous chapter especially hamper conceptual modeling. We need to come up with a more suited choreography language for the early phases of choreography design. The assessment of the existing languages has shown that both UML Communication Diagrams and BPSS lack support for a wide range of requirements. That is the reason why, unlike in the previous chapter, simply extending these languages is not an option. The fact that none of the two languages fully supports the requirement of a graphical notation underlines this argument.

Therefore, we are going to introduce new language proposals for interaction modeling on the conceptual level. Before actually diving into the languages, Section 5.1 will present a formal model that will be used throughout this chapter. The formal model is based on a special kind of labeled Petri nets.

Section 5.2 introduces Let's Dance, a novel choreography language inspired through the Service Interaction Pattern initiative. The different abstraction levels and modeling constructs are explained and the formal semantics is given through a mapping to interaction Petri nets. Finally, Let's Dance is validated and the generation of observable behavior models is studied.

Section 5.3 introduces iBPMN as extension to BPMN for interaction modeling. While reusing much of BPMN's notation, iBPMN's semantics is significantly different to that of classical BPMN. Again, a language overview is given, the formal semantics is provided based on interaction Petri nets and the generation of observable behavior models is discussed.

While interaction modeling comes with a number of advantages over interconnection model-

ing, new anomalies arise that need to be tackled. It is possible to specify choreographies where there does not exist a set of roles that collectively realize the behavior specified in the choreography. The corresponding beauty criterion for choreographies is called *realizability* and the different dimensions of it are investigated in Section 5.4.

This chapter concludes with Section 5.5 where the results are discussed. The anti-patterns from the previous chapter are also re-discussed in the context of interaction modeling.

5.1 Formal Model

This section introduces interaction Petri nets as formal model for interaction models. This enables unambiguous interpretation of choreographies and serves as basis for formal verification. Interaction Petri nets are an extension to Petri nets (cf. Section 2.4.3). They are labeled Petri nets, realizing the choreography-specific semantics through special labeling of transitions. The labeling also allows to relate choreographies to conversations. Interaction Petri nets are inspired by conversation models as presented in Section 2.4.2, considering interactions as one atomic step. Interaction Petri nets go beyond conversation models as they natively support the notion of concurrency. This is important as all choreography languages also support concurrency (cf. Section 2.3). Consequently, a more intuitive formal representation is possible using interaction Petri nets.

5.1.1 Basic Definitions

Interaction Petri nets consist of places and transitions that are connected through a flow relation. Places can contain tokens, which in turn are needed to enable transitions. Once a transition fires, tokens are consumed and produced. That way tokens flow through the net.

There are three different interpretations for transitions. They are either interpreted as (i) interactions, firing of which relates to a message exchange in a conversation, (ii) silent steps that are controlled by one or several roles, or (iii) silent steps that are not controlled by any particular role. Silent steps do not relate to message exchanges in a conversation.

Message types are first-class constructs in interaction Petri nets, allowing the distinction between, for example, acceptance and rejection messages. In the following definitions, we denote the set of all roles by R. A participant involved in a conversation plays one or several roles, for example, “bidder” or “seller”. The set of message types is denoted by MT.

Definition 5.1 (Interaction Petri Net) An *Interaction Petri net* is a tuple $(P, T, F, m_0, \mathit{final}, \lambda)$ where

- (P, T, F, m_0) is a Petri net,
- final is a finite set of valid final markings,
- T is partitioned into a set of interactions T_I, the set of controlled silent transitions T_C and the set of uncontrolled silent transitions T_U, and

- $\lambda : (T_I \rightarrow (R \times R \times MT)) \cup (T_C \rightarrow \wp(R)) \cup (T_U \rightarrow \{\tau\})$ is a labeling function where to each interaction a sender role, a receiver role and a message type is assigned and to each controlled silent transition a set of roles.

□

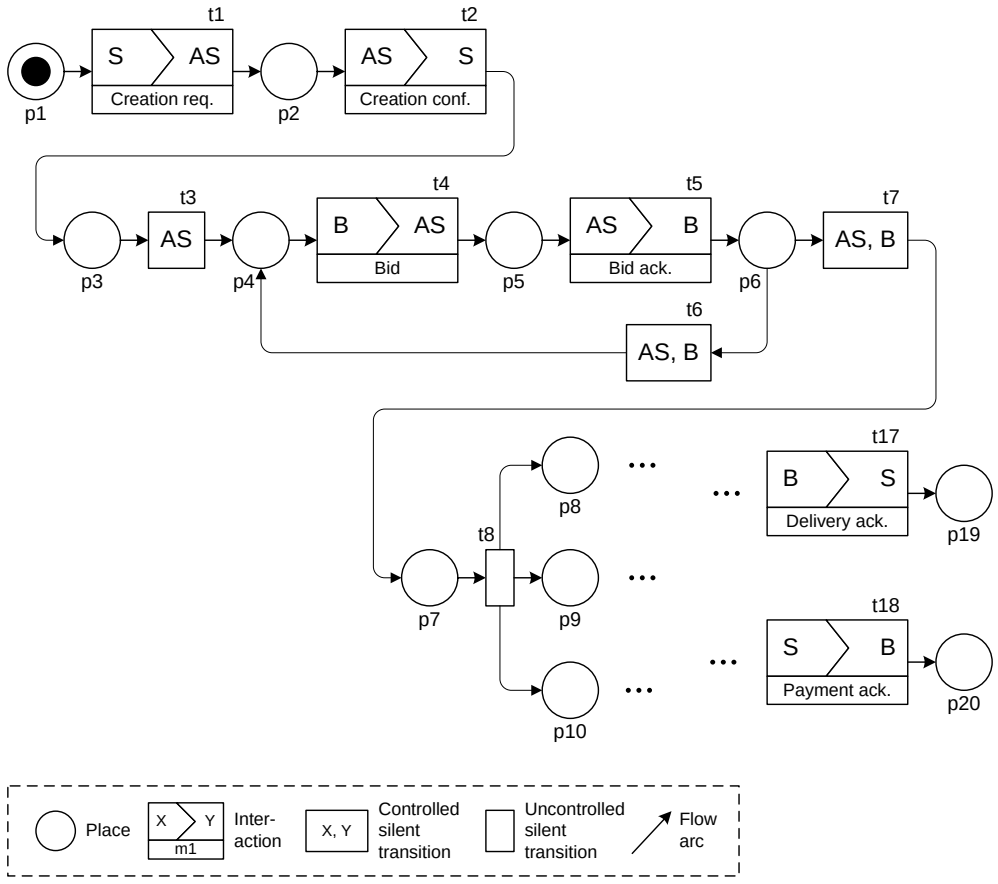


Figure 5.1: Sample interaction Petri net

Figure 5.1 depicts a sample interaction Petri net that alludes to our example from Section 1.1. The visual representation of interaction Petri nets is explained by the legend at the bottom of the figure. Places are represented by circles and interactions are represented by rectangles with three compartments, indicating the sending role (upper left corner), the receiving role (upper right corner) and the message type (bottom label). Controlled silent transitions are represented by rectangles labeled with the controlling roles, uncontrolled silent transitions are represented by empty rectangles and flow arcs are represented by arrows.

In this example, the three roles S, AS and B are involved. Transition $t1$ is an interaction with $\lambda(t1) = (S, AS, \text{'}\textit{Creation req.}\text{'})$. Transition $t6$ is a controlled silent transition with $\lambda(t6) = \{AS, B\}$ and transition $t8$ is an uncontrolled silent transition with $\lambda(t6) = \tau$. The initial marking is $[p1]$, meaning that place $p1$ contains one token and all other places are empty.

The final marking is assumed to be $[p19, p20]$. In the initial marking, only $t1$ is enabled. After firing of $t1$ the marking $[p2]$ is reached.

Definition 5.2 (Interaction Trace) An *interaction trace* is the sub-sequence of a transition sequence for a given interaction Petri net from the initial marking to a final marking that only contains the interactions of that sequence. □

A sample interaction trace for Figure 5.1 could be $\langle t1, t2, t4, t5, t4, t5, \ldots \rangle$. Silent transitions do not appear in an interaction trace. The set of interaction traces for an interaction Petri net specifies all allowed sequences of interactions, while abstracting from the silent transitions. Interaction traces operate on the model level. Therefore, we are now going to introduce the instance level to relate choreographies and conversations.

Message exchanges happen between two participants, one acting as sender and the other as receiver of a message. We denote the set of all participants by A and the set of all messages by M. Therefore, a message exchange is a triple $me = (sender, receiver, message) \in A \times A \times M$. Each message is of a particular type, given by the function $type : M \rightarrow MT$.

In order to relate choreographies and conversations, a mapping of roles to participants must be given ($map : R \rightarrow A$). As several roles might be played by the same participant, map is not necessarily injective.

Definition 5.3 (Conversation / Valid Conversation) Let a *conversation* $conv$ be a sequence of message exchanges $\langle me_1, \ldots, me_n \rangle$. $conv$ is a *valid conversation* for an interaction Petri net and a partial function $map : R \rightarrow A$ iff there exists an interaction trace $\langle t_1, \ldots, t_n \rangle$ for the interaction Petri net and for each $me_i = (s_{me}, r_{me}, m_{me})$ and $\lambda(t_i) = (s_t, r_t, m_t)$ holds $s_{me} = map(s_t)$, $r_{me} = map(r_t)$ and $type(m_{me}) = m_t$, for all $1 \leq i \leq n$. □

As the definition of valid conversations is based on interaction traces, a final marking must eventually be reached. Therefore, an interaction Petri net is not just interpreted as collection of interaction constraints, where an empty conversation or simply terminating a conversation after a few interactions would be allowed. For instance, aborting an auctioning conversation after a creation request message exchange would not be allowed. That way, interaction Petri nets are also interpreted as interaction obligations. Participants are obliged to eventually interact until the end of a conversation is reached.

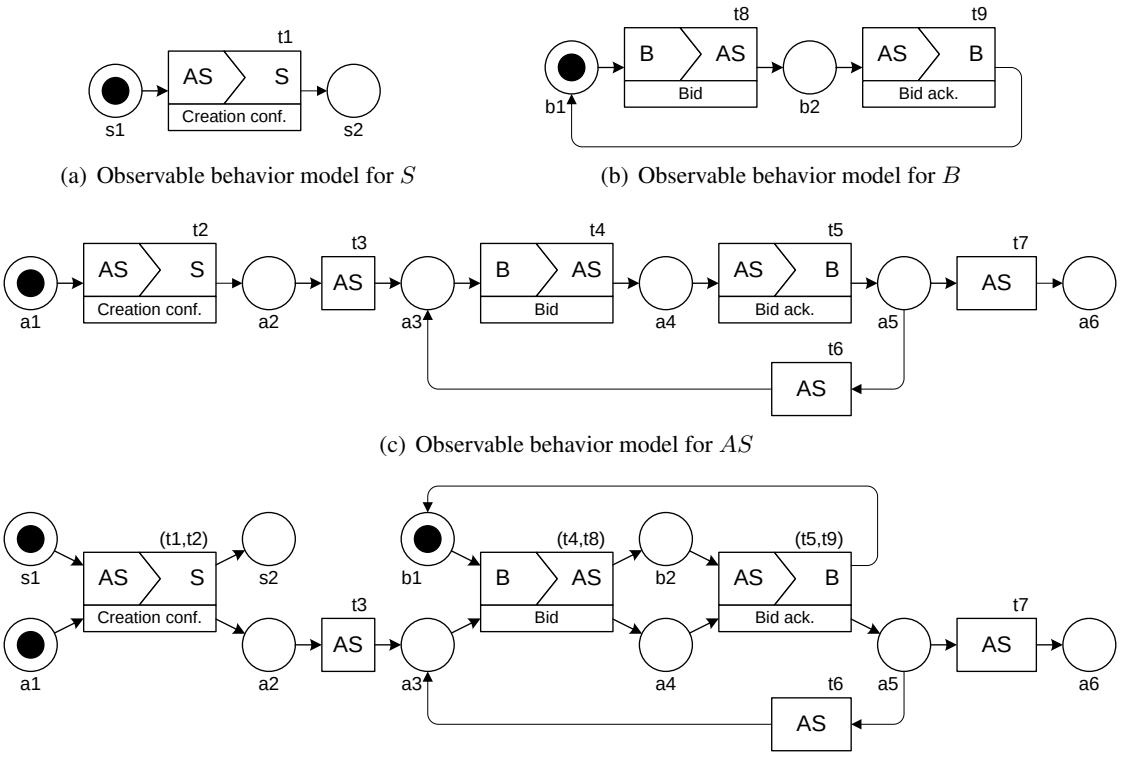
5.1.2 Composition

Interaction Petri nets are mainly used for representing choreographies. However, they can also represent observable behavior models for a role r. In this case, r must be involved in every interaction. Observable behavior models of different roles can be composed to a new interaction Petri net.

Definition 5.4 (Composability) Let $N_1 = (P_1, T_1, F_1, m_{01}, final_1, \lambda_1)$ and N_2 = (P_2, T_2, F_2, m_{02}, $final_2$, λ_2) be two disjoint interaction Petri nets representing the observable behavior models of roles r_1 and r_2. N_1 and N_2 are composable iff for each transition $t_1 \in T_1$ where r_2 is involved there is a $t_2 \in T_2$ such that $\lambda_1(t_1) = \lambda_2(t_2)$ and for each transition $t_2 \in T_2$ where r_1 is involved there is a corresponding $t_1 \in T_1$ such that $\lambda_1(t_1) = \lambda_2(t_2)$. □

We distinguish the synchronous composition and the asynchronous composition of observable behavior models. During the synchronous composition, for every pair of corresponding interactions a new interaction is created in the resulting net. The places are simply copied, the flow relationships are set accordingly and the initial marking and the valid final markings are derived.

Definition 5.5 (Synchronous composition) Let $N_1 = (P_1, T_1, F_1, m_{01}, final_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, m_{02}, final_2, \lambda_2)$ be the observable behavior models for roles r_1 and r_2 that are composable. The *synchronous composition* of N_1 and N_2, denoted as $N_1 \oplus_{sync} N_2$, is $(P_1 \cup P_2, T_{new}, F_{new}, m_{01} \oplus m_{02}, \{m_1 \oplus m_2 \mid m_1 \in final_1 \wedge m_2 \in final_2\})$, where $T_{new} = T_{C1} \cup T_{U1} \cup T_{C2} \cup T_{U2} \cup \{t_{(t_1,t_2)} \mid t_1 \in T_{I1} \wedge t_2 \in T_{I2} \wedge \lambda_1(t_1) = \lambda_2(t_2)\}$ and $F_{new} = (F_1 \cap (P_1 \cup T_{C1} \cup T_{U1})^2) \cup (F_2 \cap (P_2 \cup T_{C2} \cup T_{U2})^2) \cup \{(p, t_{(t_1,t_2)}) \mid (p, t_1) \in F_1 \vee (p, t_2) \in F_2\} \cup \{(t_{(t_1,t_2)}, p) \mid (t_1, p) \in F_1 \vee (t_2, p) \in F_2\}$. □



(a) Observable behavior model for S

(b) Observable behavior model for B

(c) Observable behavior model for AS

(d) Synchronous composition

Figure 5.2: Composing three observable behavior models to a choreography

Figure 5.2(d) illustrates the composition of three observable behavior models. The valid final markings of the observable behavior models, which are not displayed in the figure, are $\{[s2]\}$ for role S, $\{[b1]\}$ for B and $\{[a6]\}$ for AS. The resulting valid final markings for the synchronous composition are $\{[s2, a6, b1]\}$.

The synchronous composition reflects atomic interactions in the sense that message sending

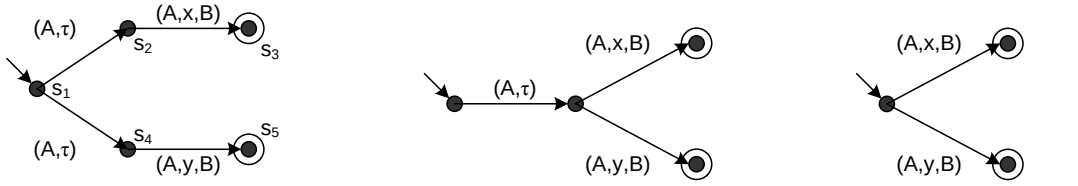
and receiving must happen at the same time. Message sending is only possible if a corresponding message receive can happen in another role. In order to reflect asynchronous communication, we can expand interaction Petri nets to open nets. A communication place and corresponding flow relationships are introduced for every label (s, r, mt) in an observable behavior model. The initial marking and the final marking can be reused as is. Finally, the open composition of open nets can be applied (cf. Section 2.4.3).

5.1.3 Role Projection

An observable behavior model must be derived from the choreography for every role. This calls for *role projection*, where those interactions are removed where the role under investigation is not involved in. In order to approach this issue, we will first use conversation models (cf. 2.4.2) and then shift the results to interaction Petri nets.

We extend the conversation models from [113] with silent transitions. A silent transition is always performed by one role only and is most commonly used for representing the decision ownership of a role. (A, τ) denotes a silent transition performed by role A.

Role projection is straightforward if the branching structures of the choreographies are not of importance. In this case, the irrelevant interactions can simply be removed and the remaining transitions set accordingly. Furthermore, the derivation of a minimal deterministic state machine would then guarantee that the desired traces can be produced. This approach is followed in [113].



(a) Conversation model

(b) Deterministic role projection for A

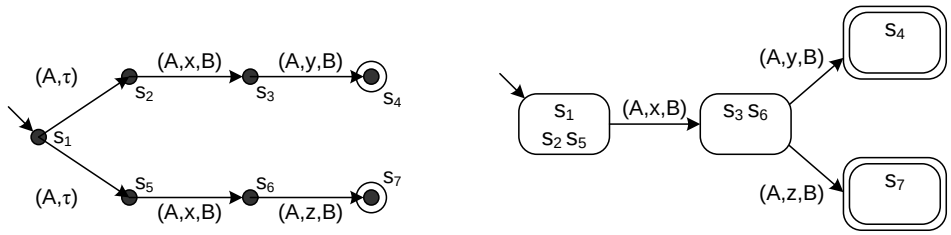
(c) Deterministic role projection for B

Figure 5.3: Deterministic role projections

Figure 5.3(a) illustrates an example where role A does an internal choice. In an interaction Petri net this would correspond to silent transitions controlled by A. Depending on this internal choice, A will either send a message x or a message y to B. A role projection producing a deterministic state machine would look as shown in Figure 5.3(b). The information that A decides first and then sends a message, i.e. it is A who decides what message is sent and not B, gets lost. After the role projections (and under the assumption of a synchronous composition), both A and B have equal influence on what message will be sent.

We argue that the ownership of choice is essential in choreographies. It makes a big difference whether A decides alone or if A and B have equal influence. This argumentation goes along with the requirement *R2* from Section 3.2. Therefore, we are going to present role projection that preserves branching structures and hence respects the moment and location of choice.

We construct the role behavior for a role r in a similar way like the operating guidelines approach calculates a most permissive partner [152]. The states of an observable behavior model can be derived by identifying all possible states in the choreography that can be reached without involvement of r. Every time an observable transition happens, it can be derived what knowledge about the current choreography state r has. Those situations are particularly interesting where r takes part in an interaction but cannot be sure about what state the choreography is in.



(a) Conversation model (b) Role projection for B

Figure 5.4: Role projection respecting branching structures

Figure 5.4(a) illustrates an example where B cannot know the choreography state when receiving message x from role A, it could be either $s3$ or $s6$. Only after the receipt of the second message y or z, B knows whether $s4$ or $s7$ was reached.

The following recursive algorithm defines role projection for a role r and a conversation model with states S, initial state s_0, final states $final$ and transition relation δ. A conversation model with new states will be created. We denote the set of all new states with N. The functions $entry, reach : N \rightarrow \wp(S)$ will be used in the algorithm. As auxiliary notation we use $s \xrightarrow{*} s'$ for denoting that there exists a (potentially empty) sequence of transitions without involvement of r from s to s'.

1. Initialize $entry := \emptyset$ and $reach := \emptyset$.

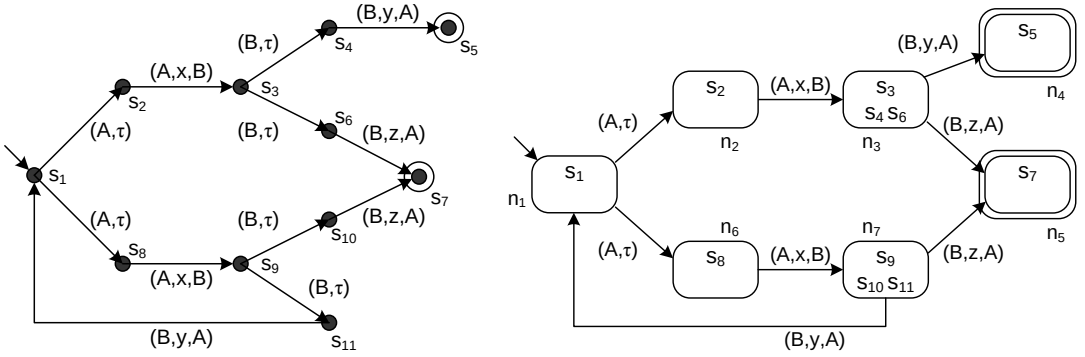
2. Create new state n and set $entry(n) := \{s_0\}$.

3. Determine the set of states that are reachable from some $s \in entry(n)$ without involvement of r, i.e. $reach(n) := \{s' \mid \exists s \in entry(n)\ (s \xrightarrow{*} s')\}$.

4. For every message m where $r \in \{send(m), recv(m)\}$ determine all states $\{s_1, \ldots, s_n\} \subseteq reach(n)$ where there exists a state s_i' such that $(s_i, m, s_i') \in \delta$.

   (a) For every $\{s_1', \ldots, s_n'\}$ such that $(s_i, m, s_i') \in \delta$ for all i find a state n' such that $entry(n') = \{s_1', \ldots, s_n'\}$. If such a state exists then add transition (n, m, n'). Otherwise create a new state n', set $entry(n') := \{s_1', \ldots, s_n'\}$, add transition (n, m, n') and recursively proceed with step 3 where $n := n'$.

5. Determine the initial state n_0 where $entry(n_0) = \{s_0\}$. Determine all final states n where $final \cap reach(n) \neq \emptyset$.



(a) (b)

Figure 5.5: Conversation model and role projection for A

The conversation model in Figure 5.4(b) results from applying the role projection algorithm on the conversation model in Figure 5.4(a) for role B. Figure 5.5(a) illustrates a more complex choreography and Figure 5.5(b) the role projection for role A. First, state n_1 is created. $entry(n_1)$ is set to $\{s_1\}$. $reach(n_1)$ equals $entry(n_1)$ as no other state is reachable from s_1 without involvement of A. States n_2 and n_6 are produced as s_2 and s_8 serve as s_1' in step 4(a) of the algorithm. In state n_3, A cannot know whether the overall choreography is in state s_3, s_4 or s_6 ($entry(n_3) = \{s_3\}$ and $reach(n_3) = \{s_3, s_4, s_6\}$). Only after the next interaction with B, A knows what final state was reached. Similarly, A knows in n_7 that the overall choreography must be in state s_9, s_{10} or s_{11}.

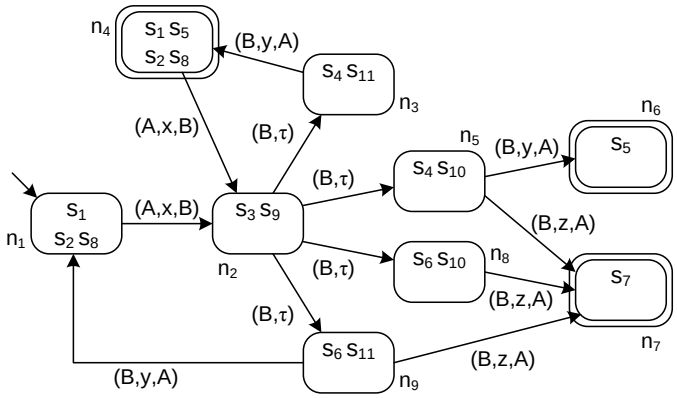


Figure 5.6: Role projection for B

The role projection for role B looks a bit more complicated. Already in the start state n_1, B does not know whether the overall choreography is in state s_1, s_2 or s_8 as B cannot observe the choice made by A. After having received message x from A, B still does not know which path was taken by A. Even worse, B can now make a decision himself. However, when starting from s_3, either s_4 or s_6 could be reached. When starting from s_9, either s_{10} or s_{11} could be reached.

Therefore, all possible combinations are enumerated in the state space. When reaching n_5, the next interaction with A brings clarity about the state of the overall choreography. When starting from n_8, only a z message can be sent. When starting from n_3 it is particularly interesting to see that B cannot know whether s_1 or s_5 was reached in the overall choreography and whether the end of the choreography was reached or not.

The algorithm for role projection produces new states N that are not present in the original set of states S. Sometimes, however, it is possible to reuse S, i.e. $N \subseteq S$. This is the case if $entry(n)$ contains at most one state s for each $n \in N$. Here, s could be reused as n. The necessary and sufficient condition for $|start(n)| \leq 1$ is that for all reachable $s_1, s_2, s_2', s_3, s_3' \in S$ and $t \in T$ for which holds $s_2 \neq s_3$, $s_1 \xrightarrow{*} s_2' \xrightarrow{t} s_2$ and $s_1 \xrightarrow{*} s_3' \xrightarrow{t} s_3$ this implies that $s_2' = s_3'$. Informally speaking, if two states s_2 and s_3 are reachable from s_1 via sequences of non-observable transitions and one observable transition t this implies that the role can choose (and therefore know) whether it is in s_2 or s_3 due to the fact that the role "owns" the branching decision.

This special case is interesting as the role projection can be done via pure structural rewriting of the conversation model. Any sequence $s_1 \xrightarrow{*} s_2' \xrightarrow{t} s_3$ can be replaced by $s_1 \xrightarrow{t} s_2$. Figure 5.5(b) shows an example where such structural rewriting equals performing the role projection. The example from Figure 5.6 shows an example where this is not the case due to the decision made by A which is not immediately communicated to B. As decisions are immediately communicated in most choreographies, structural rewriting can be applied for a wide range of choreographies.

So far, we have only considered role projection in conversation models. In order to shift this approach back to interaction Petri nets, one could follow the following strategy: (1) Given a bounded interaction Petri net, the corresponding conversation model is computed. (2) The role projections are generated. (3) The role projections are translated back to interaction Petri nets.

Step (3) would produce interaction Petri nets where there is exactly one token in every reachable marking. As an optimization, tools such as Petrify [60] could be used to reconstruct concurrency. However, this approach often does not preserve the initial Petri net structure. The resulting net might look completely different to the original net. Therefore, we are now going to investigate a transformational approach where the original Petri net structure is preserved to a larger extent.

Preserving structure is especially easy in those cases where role projection can be done through structural rewriting. We have seen that we simply need to replace pairs of transitions by one transition. In an interaction Petri net we need to introduce transitions that behave like firing the two original transitions in a direct sequence.

Definition 5.6 (Transition aggregation) Given an interaction Petri net $(P, T, F, m_0, final, \lambda)$ and two transitions $t_1, t_2 \in T$, then the *transition aggregation* of t_1 and t_2 is a new transition $t_{1,2}$ where $\bullet t_{1,2} = \bullet t_1 \cup (\bullet t_2 \setminus t_1\bullet)$ and $t_{1,2}\bullet = t_2 \bullet \cup (t_1 \bullet \setminus \bullet t_2)$. □

Figure 5.7 illustrates the definition. Here, t_1 and t_2 are aggregated and t_3 is created.

Interaction Petri nets do not have the concept of arc weights that would allow to consume

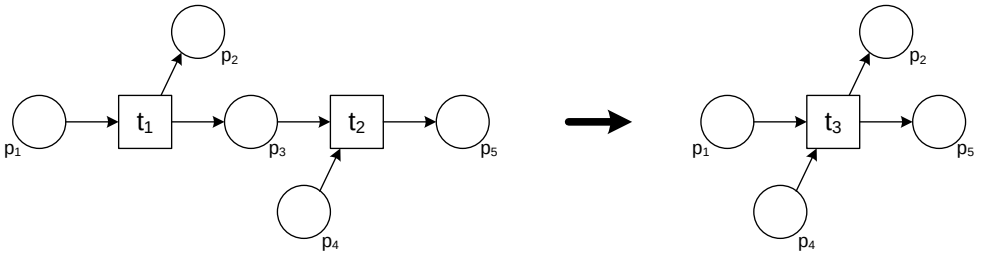
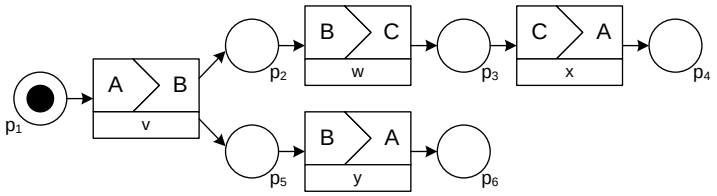


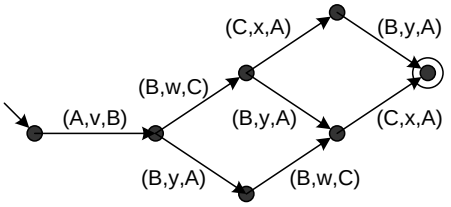
Figure 5.7: Transition aggregation

multiple tokens from one place or producing multiple tokens onto one place. A sufficient (but not necessary) structural criterion for transition aggregation to work properly is $\bullet t_1 \cap \bullet t_2 \subseteq t_1\bullet$ and $t_1\bullet \cap t_2\bullet \subseteq \bullet t_2$. This guarantees that at most one token must be consumed from or produced onto each place.

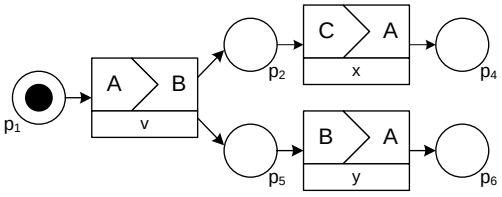
Due to the concept of concurrency it might be the case that one transition in an interaction Petri net corresponds to multiple transitions in the conversation model. Figures 5.8(a) and 5.8(b) illustrate this.



(a) Interaction Petri net



(b) Conversation model

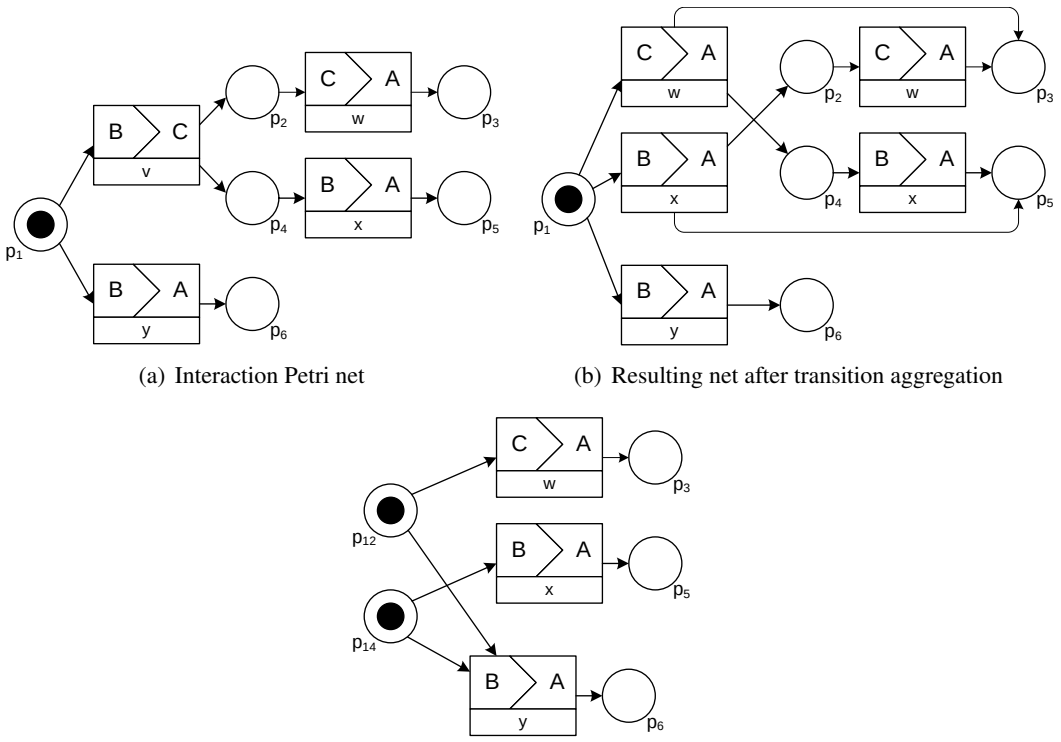


(c) Resulting net after transition aggregation

Figure 5.8: Concurrency in interaction Petri nets and conversation models

The definition of transition aggregation respects the notion of concurrency to a large extent. Only places that are input or output of the affected transitions are touched. All flow relationships concerning other places are unaffected by the transition aggregation. Therefore, transition aggregation for the example from Figure 5.8(a) results in the interaction Petri net displayed in Figure 5.8(c). Here, place p_3 has already been removed as it is never marked.

While transition aggregation already keeps the original structure to a large extent, it might lead to undesirable results in certain scenarios. Figure 5.9(a) illustrates an example, where transition aggregation leads to duplicate transitions in the resulting interaction Petri net (Figure 5.9(b)). The interaction Petri net in Figure 5.9(c) consists of less nodes and shows equivalent



(a) Interaction Petri net

(b) Resulting net after transition aggregation

(c) Resulting net after place aggregation

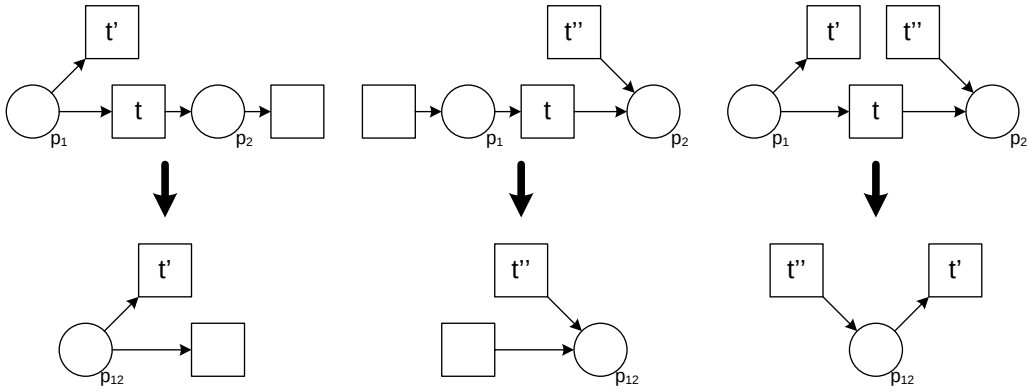
Figure 5.9: Optimized results through the place aggregation strategy

behavior with the previous result. The result from Figure 5.9(c) can be achieved when applying a *place aggregation* strategy. Instead of creating additional transitions, places are replaced by new places.

Definition 5.7 (Place aggregation) Given an interaction Petri net $(P, T, F, m_0, \mathit{final}, \lambda)$ and a transition $t \in T$, then the *place aggregation* for t results in new places $p_{i,j}$ that are created for each pair $p_i \in {\bullet}t$, $p_j \in t{\bullet}$ and flow relationships $(p_{i,j}, t')$ for all $(p_i, t'), (p_j, t') \in F$ and $(t', p_{i,j})$ for all $(t', p_i), (t', p_j) \in F$. t and places ${\bullet}t \cup t{\bullet}$ are removed. □

A structural condition for applying place aggregation for a transition t is that ${\bullet}t \cap t{\bullet} = \emptyset$ and either there is no $t' \neq t$ sharing an input place with t or there is no $t'' \neq t$ sharing an output place with t. This is a sufficient condition for ensuring that place aggregation does not allow transition sequences that were not possible in the original interaction Petri net. Figure 5.10 illustrates three examples.

The two aggregation rules with their structural constraints can be used for generating role projections of choreographies given as interaction Petri nets. However, merely operating on the structural level and applying the rules as long as no non-observable interactions are left in the net entails the danger of non-termination of the algorithm. It must be considered that if two



(a) Allowed (b) Allowed (c) Not allowed

Figure 5.10: Optimized results through the place aggregation strategy

reachable transitions t_1, t_2, where some output place of t_1 is an input place of t_2, does not imply that there exists a reachable marking m such that $m \xrightarrow{t_1} m' \xrightarrow{t_2} m''$. Therefore, a newly introduced transition $t_{1,2}$ would not be reachable. In some cyclic nets an infinite number of unreachable transitions could be generated.

5.2 Let's Dance

Given the poor support for many Service Interaction Patterns by the interaction modeling languages UML Communication Diagrams and BPSS, as well as the lack of graphical notation in the case of WS-CDL, Let's Dance is introduced as novel language targeted at choreography modeling on the conceptual level. It was first introduced by Zaha et al. [220] and further developed in the course of a joint choreography project between SAP Research and the Queensland University of Technology in 2006. Let's Dance is specifically designed to support a wide range of Workflow and Service Interaction Patterns. Being a conceptual language, message formats and data flow are neglected. Specification of behavioral dependencies is the main focus of Let's Dance. However, Let's Dance comes with a second diagram type, called *role-based view*.

5.2.1 High-level Choreographies

We can distinguish between *role-based views* on choreographies and *milestone views*. Let's Dance directly supports both of these views. First, we are going to take a look at how Let's Dance can be used to model roles, participants and their relationships. Figure 5.11 gives an overview of the different language constructs that are available for this purpose.

At the center of attention we find collaborating *roles* (depicted as boxes). We normally assume that there is at most one participant involved in a choreography per role. If potentially multiple participants of the same role, e.g. logistics companies in the role "Shipper", take part in a choreography, overlaid boxes are used to represent such multiplicity (e.g. role *B*).

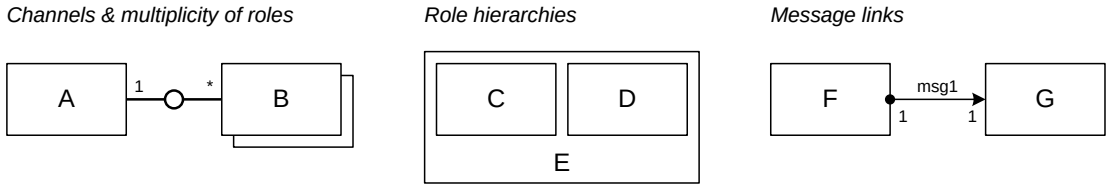


Figure 5.11: Language constructs for role-based views in Let's Dance

Channels (depicted as small circles on lines) represent interaction dependencies between different roles, e.g. between *A* and *B*. The meaning of such a channel is that participants of roles that are connected through channels might interact. If there is no channel between roles "Buyer" and "Seller" then a buyer cannot interact with a seller.

Containment of boxes depicts *role hierarchies*, e.g. between sub-roles *C* and *D* and super-role *E*. Such hierarchies can have one of the following meanings: (1) One role is *part of* another role. In this case a participant of the sub-role is part of the participant of the super-role. Imagine e.g. a car manufacturer that can be decomposed into different production sites, warehouses and inventory management. In terms of interaction behavior, an interaction assigned to the super-role would be assigned to one or several out of the sub-roles in a refinement step. For instance, a second company interacting with the car manufacturer might in fact interact with the inventory management only. (2) A sub-role could also be a *specialization* of the super-role. The sub-roles therefore inherit the interaction behavior of the super-role. If an interaction is possible with a participant of the super-role then such interaction must also be possible with participants of all sub-roles. As an example for role specialization we could deal with a super-role "Carrier" and its specializations "Land Carrier" and "Air Carrier".

Channels can be refined into *message links*. While channels are not directed, message links describe the flow of a message from a sender to a receiver, e.g. from role *F* to *G*. Message names need to be specified for every link, e.g. *msg1* in Figure 5.11. As an example, imagine a channel representing that a seller interacts with an auctioning service. Different message links could then refine this channel into an auction creation request link from the seller to the auctioning service and a creation confirmation flowing back.

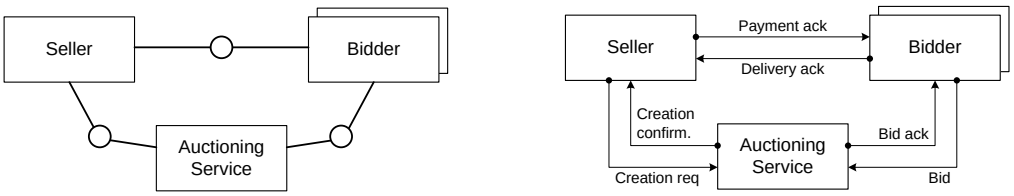


Figure 5.12: Role-based view in Let's Dance

Figure 5.12 shows the role-based view of our auctioning scenario. The three roles Seller, Bidder and Auctioning Service are interconnected through channels (in the left diagram). Furthermore, it is represented that at most one seller, at most one auctioning service and potentially many sellers participate in a conversation. The right diagram shows the refinement: the channels

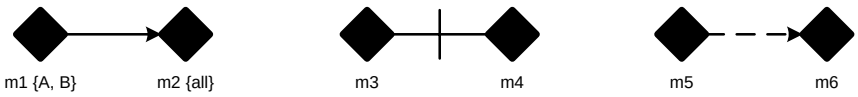


Figure 5.13: Language constructs for milestone views in Let's Dance

are refined through message links.

As a second artifact type for high-level choreographies, *milestone models* are available. While role-based models only capture structural aspects of a choreography, milestone models show high-level behavioral dependencies. Figure 5.13 introduces all language constructs and shapes for milestone models.

As central concept we find *milestones* (depicted as diamonds). They represent goals and sub-goals that are to be reached during the choreography. Different control flow constructs show the dependencies between the milestones. At this point we will only explain the semantics of *Precedes* relationships. The other relationship types will be introduced in section 5.2.2. If a milestone m_1 precedes a second milestone m_2 then m_1 has to be reached before m_2 can be reached. However, there is no guarantee that m_2 will eventually be reached if m_1 was reached.

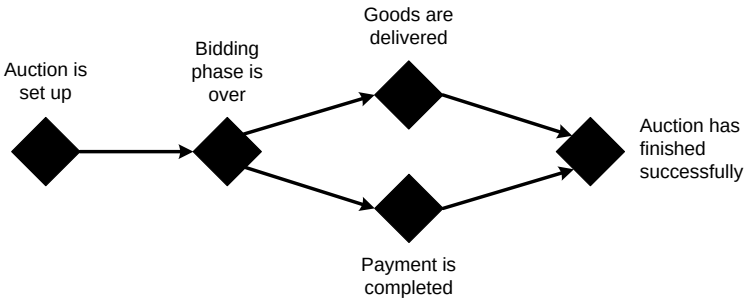


Figure 5.14: Milestone view in Let's Dance

Figure 5.14 shows a milestone diagram for the auctioning scenario. The different milestones are connected through *Precedes* relationships. This indicates that e.g. the bidding phase must be over before the payment can be completed. However, it is not said that the payment must eventually be completed if the bidding phase has finished.

5.2.2 Interaction Modeling

The main focus of Let's Dance is to capture interactions and their behavioral dependencies. An elementary interaction is a combination of a send activity and a receive activity. An actor reference belonging to a role is given for every activity. This reference indicates which activity instances must be performed by the same participant. Typically, there is only one participant per role involved in a conversation. In these cases the actor reference can be omitted in the diagrams.

Figure 5.15 shows the interactions leading to the "auction is set up" milestone in the auction example. As already seen in the milestone example, *Precedes* relationships between two interactions indicate that an instance of the target interaction can only happen if the instance

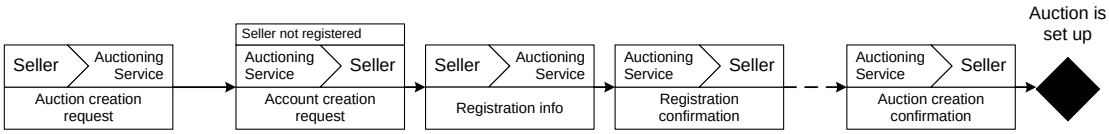


Figure 5.15: Interaction modeling in Let's Dance

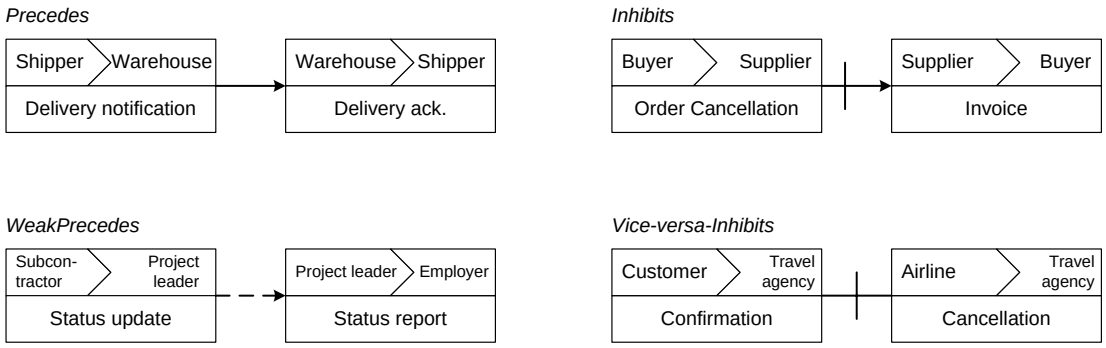


Figure 5.16: Basic control flow constructs in Let's Dance

of the source interaction has already happened. Imagine a logistics example where a delivery acknowledgment should only be sent after a delivery notification. An *Inhibits* relationship indicates that an instance of the target interaction can only happen if no instance of the source interaction has happened yet. In an order example an invoice should not be sent after an order cancellation by the buyer. Scenarios where two interactions inhibit each other, i.e. an instance of either one or the other interaction can complete, are very common. Consider e.g. a travel agency that either receives a confirmation from the customer or a cancellation from the airline when the reserved seats become unavailable. Therefore, a special notational element for *Vice-Versa-Inhibits* is introduced. A *WeakPrecedes* relationship means that an instance of the target interaction can only happen after the instance of the source interaction has already completed or was skipped. Imagine a project management scenario where the project leader expects status updates from a subcontractor that are merged into a status report for the employer. However, in special cases the project leader and the subcontractors can agree that no status update is needed. The three examples for the different relationship types are depicted in Figure 5.16.

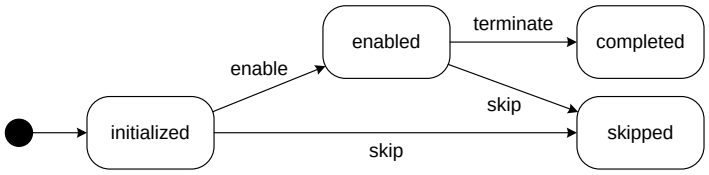


Figure 5.17: Interaction instance life cycle in Let's Dance

Interaction instances can be in the states *initialized*, *enabled*, *completed* and *skipped* (cf. Figure 5.17). An interaction instance becomes skipped if any of the inhibiting instances has

completed. An interaction instance becomes enabled if there are no *Precedes* and *WeakPrecedes* relationships targeting the corresponding interaction or all preceding instances are completed and all weakly preceding instances are completed or skipped. An instance must only execute, i.e. the actual message exchange occurs, if it is enabled. After the execution the instance is in the state completed. In the case of skipping, we find "dead path elimination" execution semantics. This means that following instances are skipped along the *Precedes* relationships. This semantics is the reason why *Precedes* and *WeakPrecedes* relationships must not occur in cycles.

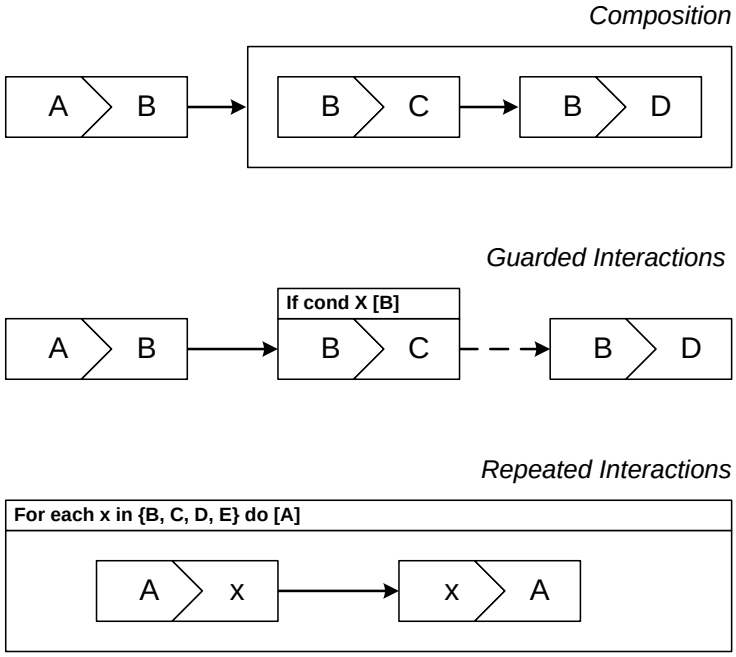


Figure 5.18: Advanced control flow constructs in Let's Dance

In addition to the three relationship types, there are further control flow constructs in Let's Dance. Several interactions can belong to a *composite interaction*. None of the contained interaction instances can become enabled before the enclosing composite interaction instance has become enabled and the composite interaction instance can only complete after all contained interaction instances have completed or have been skipped. Interactions can also be guarded, i.e. at the moment an interaction instance could become enabled, a certain constraint must be fulfilled. If this constraint is not fulfilled the instance is skipped. Finally, repetitions and parallel branching with an unbounded number of branches are modeled through repeated interactions. There are four types of repeated interactions in Let's Dance: "while", "repeat", "for each (sequential)" and "for each (concurrent)". "For each" repetitions have an expression attached that determines a collection over which the repetition is performed. The knowledge about how many instances are to be created for this interaction might be available at design-time or might only be known at runtime. Repetitions can have stop conditions attached to it. E.g. a repeated receive interaction should be stopped as soon as answers from 10 participants have arrived. The expressions attached to guarded and repeated interactions can be written in plain English, as

Let's Dance is not tied to any specific expression language. However, it must be defined which actor is going to check whether a condition evaluates to true or to which collection results from a repetition expression.

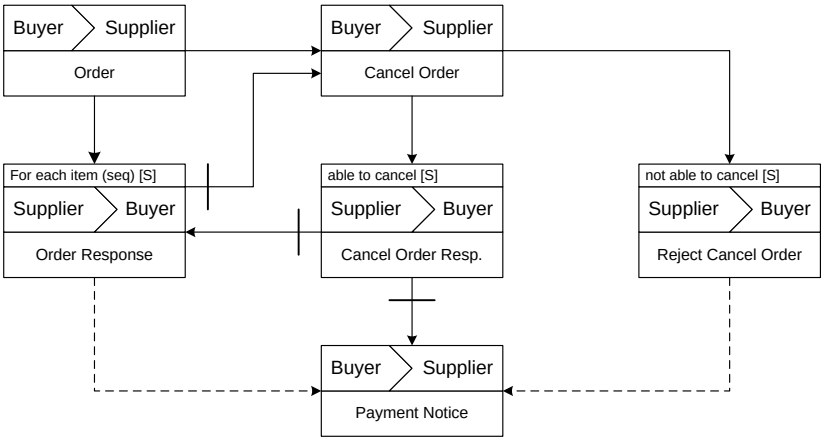


Figure 5.19: Sample interaction model in Let's Dance

Figure 5.19 shows a sample interaction model from the order management domain in Let's Dance. After a buyer has submitted her order to a supplier, the supplier sends back an order response for every item in the order. As soon as all responses have been sent, the order cannot be canceled by the buyer any longer. This is expressed through the *Inhibits* relationship from "Order Response" to "Cancel Order". If a cancellation is issued by the buyer on time, the supplier can decide whether it can still be canceled or not. If cancellation is still possible, the remaining order responses are skipped and the buyer does not need to pay. In the case where cancellation is not possible, a corresponding rejection notice is sent to the buyer and the buyer has to pay for the order. The buyer notifies the supplier through a payment notice.

5.2.3 Formal Semantics

This section first presents the meta-model for Let's Dance models. The execution semantics of interaction models will then be given through a translation to interaction Petri nets.

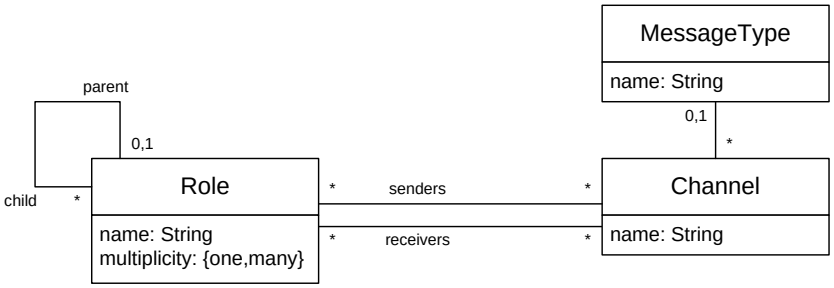


Figure 5.20: Meta-model for role-based models

Roles have a multiplicity attribute indicating whether at most one participant of that role is involved or whether potentially many participants can take part in the same conversation. Roles have parent / child relationships and they are connected through channels. Each channel has at most one message type assigned. Figure 5.20 illustrates this using the UML class diagram notation [5].

We introduce a unified meta-model for milestone, scenario and interaction models. It is illustrated in Figure 5.21. Interactions and milestones are flow objects that are connected through relationships of types $Precedes$, $WeakPrecedes$ or $Inhibits$. Interactions can be repeated and can have a guard condition attached. We distinguish elementary interactions and complex interactions. Elementary interactions have a sender role, a receiver role and a message type defined. A complex interaction can contain flow objects.

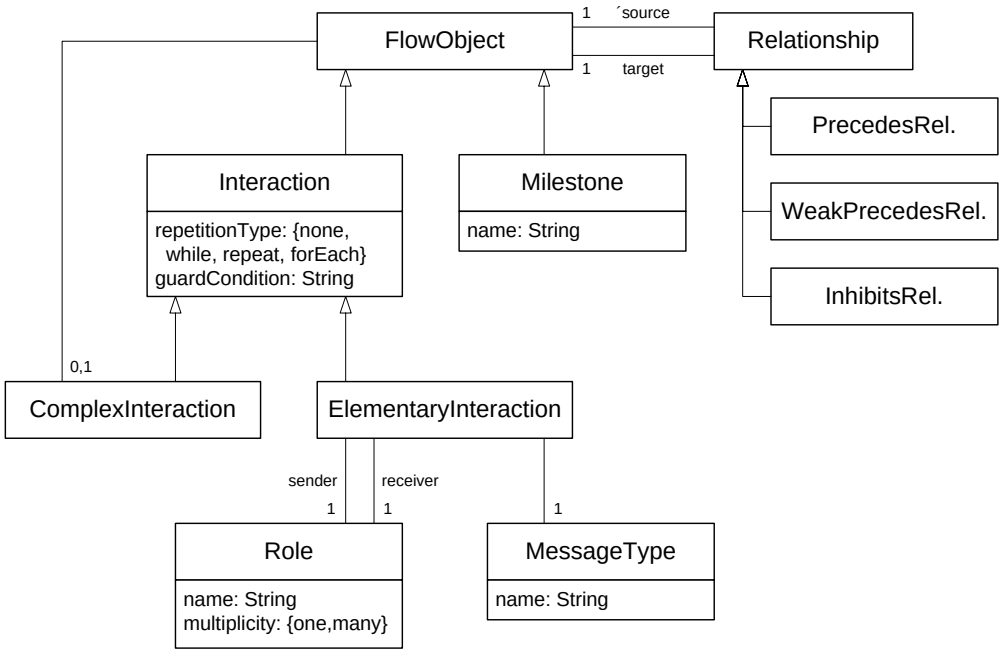


Figure 5.21: Meta-model for interaction models

Certain syntactic constraints have to be met.

- No relation that starts inside a repeated (composite) interaction crosses the boundary of this interaction.
- There are no precedence, i.e. $Precedes$ or $WeakPrecedes$, relationships between ancestors and descendants.
- There are no cyclic precedence dependencies.

All Let's Dance choreographies without for-each constructs can be mapped to 1-safe interaction Petri nets. 1-safe means that in any reachable state there is at most one token in each

place. Concurrent for-each cannot be represented properly in interaction Petri nets. The general idea behind the mapping can be described as follows:

- Each interaction gets an "interface" consisting of a number of places tokens can be consumed from and tokens can be produced to. The tokens residing on the input interface places determine the enablement of the interaction. In this context, each $Precedes$ relationship pr is represented by two places, $p_{(pr,do)}$ for positive enablement (the previous interaction completed successfully) and $p_{(pr,skip)}$ for negative enablement (the previous interaction was skipped). Each $WeakPrecedes$ relationship wp is represented by one place $p_{(wp,do)}$. Each $Inhibits$ relationship inh is represented by two places $p_{(inh,nok)}$ and $p_{(inh,ok)}$. Here, at any point in time there is one token on one of the two places. $p_{(inh,nok)}$ represents that the interaction was inhibited and $p_{(inh,ok)}$ that is was not inhibited.

  Regarding the output places, successful completion of an interaction leads to the production of tokens onto $p_{(pr,do)}$ and $p_{(wp,do)}$. Should there be outgoing $Inhibits$ relationships, it is ensured that $p_{(inh,nok)}$ is marked. Therefore, n^2 transitions are needed where n is the number of outgoing $Inhibits$ relationships.

- Skipping of interactions is done through skip transitions. There are different scenarios where skipping applies. (i) At least one of the $p_{(pr,skip)}$ places contains a token. (ii) $p_{(inh,nok)}$ contains a token. (iii) The guard condition evaluates to false.

  Figure 5.22(a) shows an interaction with two incoming $Precedes$ relationships, one outgoing $Precedes$ and one outgoing $WeakPrecedes$ relationship. In Figure 5.22(b), the interaction transition is only enabled if both $p_{(pr_1,do)}$ and $p_{(pr_2,do)}$ are marked. $p_{(pr_1,skip)}$ or $p_{(pr_2,skip)}$ being marked leads to skipping the interaction.

  Figure 5.22(c) shows an interaction with an incoming $WeakPrecedes$, an incoming $Precedes$, an incoming $Inhibits$ and an outgoing $Precedes$ relationship. In Figure 5.22(d), the interaction transition is only enabled if $p_{(wp_1,do)}$, $p_{(pr_1,do)}$ and $p_{(inh_1,ok)}$ are marked. If $p_{(inh_1,nok)}$ is marked the upper skip transition will fire, or if $p_{(pr_1,skip)}$ is marked the lower skip transition will fire.

  Figure 5.22(e) shows an interaction with an outgoing $Inhibits$ relationship and a guard condition attached. In Figure 5.22(f), two interaction transitions are used for distinguishing the case that $p_{(inh_1,ok)}$ is marked from the case that $p_{(inh_1,nok)}$ is marked. In the first case, place $p_{(inh_1,nok)}$ will be marked, otherwise $p_{(inh_1,nok)}$ remains marked.

- All interactions that are not contained in a complex interaction and do not have incoming $Precedes$ or $WeakPrecedes$ relationships are enabled from the start. This must be considered in the initial marking.

- Complex interactions are represented using a start transition enabling all interactions without incoming $Precedes$ or $WeakPrecedes$ relationships and an end transition that is enabled as soon as all interactions have completed or have been skipped.

  Figure 5.22(g) illustrates a complex interaction containing two elementary interactions. The corresponding interaction Petri net mapping in Figure 5.22(h) shows that the succeeding interactions are enabled even if the contained interactions are all skipped.

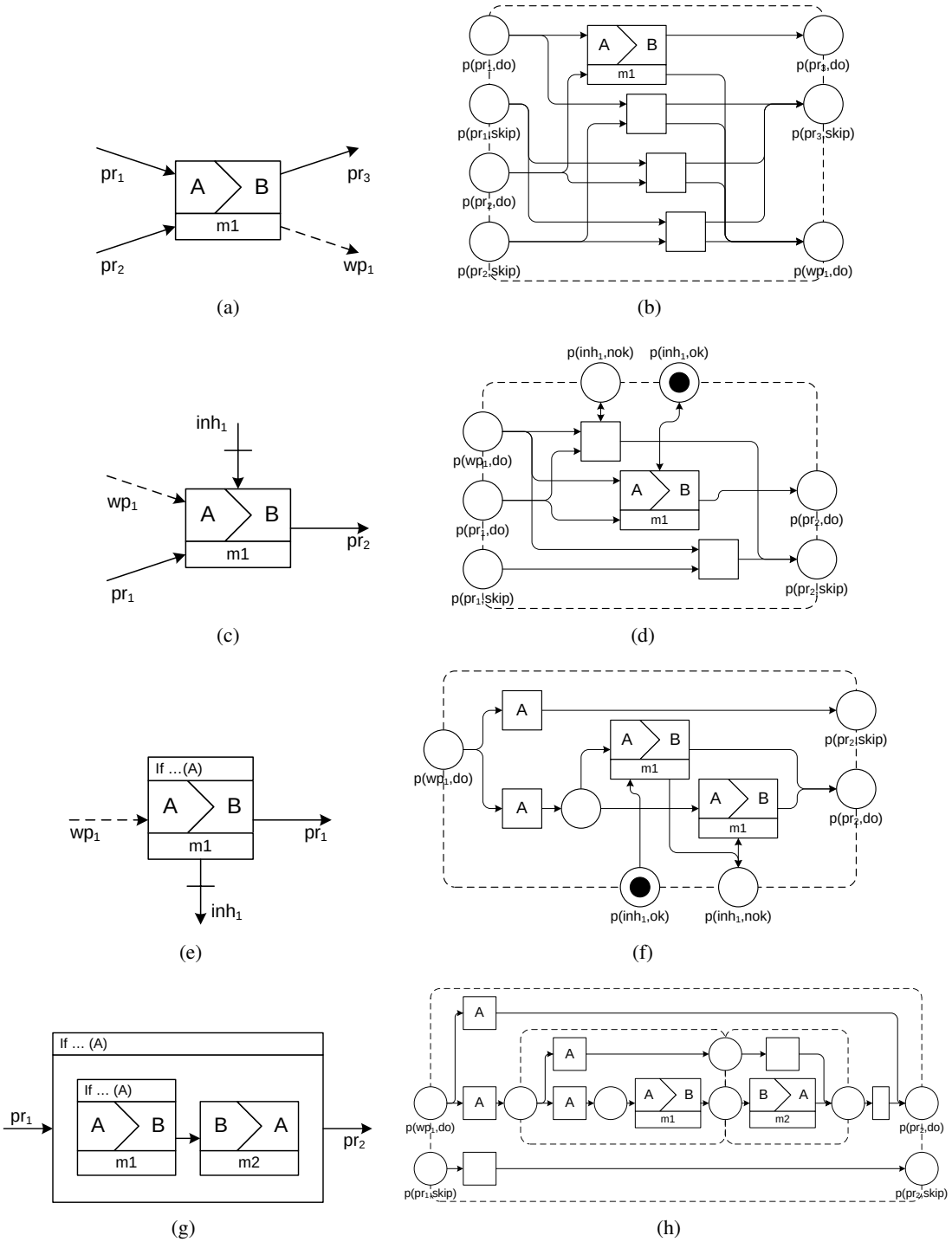


Figure 5.22: Let's Dance models and their interaction Petri net counterparts

- Repetitions have the special challenge that all places representing $Inhibits$ relationships must be reset to the initial marking, i.e. all $p_{(inh,nok)}$ contained in a complex interaction must be emptied and all $p_{(inh,ok)}$ must be marked.

- $Inhibits$ relationships targeting complex interactions have the notion of cancellation. This is realized by reading from places $p_{(inh,ok)}$ and $p_{(inh,nok)}$ for every elementary interaction contained in the complex interaction and corresponding skip transitions.

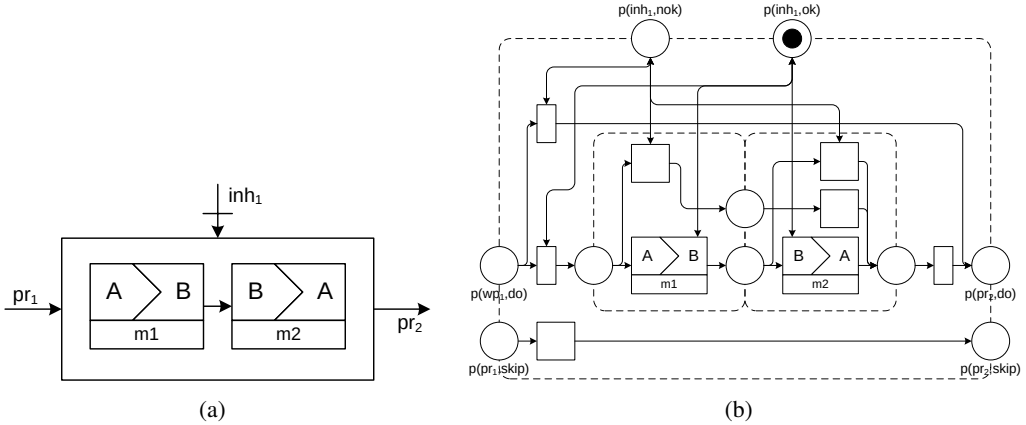


Figure 5.23: Canceling nature of $Inhibits$ relationships

Figure 5.23(a) illustrates a complex interaction with an incoming $Inhibits$ relationship. In Figure 5.23(b), the two interaction transitions also read from $p_{(inh_1,ok)}$ for ensuring that they can only fire as long as the surrounding complex interaction was not inhibited.

5.2.4 Validation

Let's Dance was designed by the same research group who previously identified the Service Interaction Patterns. Therefore, it is no surprise that most patterns are directly supported in this language. Many of the requirements from Chapter 3 are supported, too. A graphical notation is present (Requirement *C1*). A structural view is provided through a separate diagram type (*C2*). Decomposition of interactions is supported through complex interactions (*C4*). However, modularity is not given as there is no mechanism for referring to elements in other diagrams (*C3*). That way, reusability of subchoreographies is also hampered (*C5*).

An unlimited number of roles can be present in one choreography (*R1*). Ownership of choices can be defined (*R2*) and multiplicity of roles is also supported (*R3*). Participant reference passing was mentioned in different publications presenting the language, but always neglected in formal semantics (*R4*). Cancellation is possible through the $Inhibits$ relationship (*R5*). Time constraints could be realized through a special interactions that are interpreted as arm timer and expiration (*R6*). Again, this feature was never followed up on in detail and largely neglected in this section as well.

5.2.5 Generating Observable Behavior Models

For every role in a choreography an observable behavior model can be generated. The basic idea is to reduce the choreography until no interaction is left, where the role in question is not involved in. These interactions are marked as τ-interactions and then reduction rules apply to the choreography. Figure 5.24 illustrates the rules.

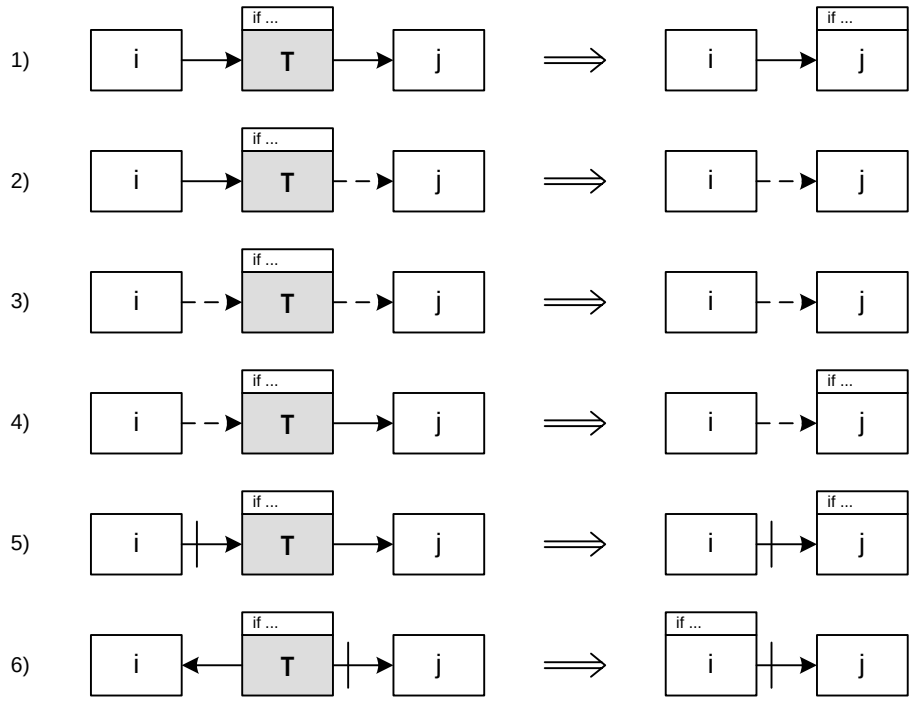


Figure 5.24: Reduction as part of the generation of observable behavior models

A combination of two $Precedes$ relationships is reduced by a new $Precedes$ relationship. A combination of a $Precedes$ and a $WeakPrecedes$ relationship or a combination of two $WeakPrecedes$ relationships is reduced by a new $WeakPrecedes$ relationship. A combination of an $Inhibits$ and a $Precedes$ relationship is reduced by a new $Inhibits$ relationship. The full formal description of the algorithm for generating observable behavior models can be found in [220].

5.3 iBPMN

Let's Dance can be considered a first academic proposal for a graphical language for modeling choreographies on the conceptual level. While covering more Service Interaction Patterns than any choreography language before, Let's Dance did not succeed to be adopted in practice. The language was used in several research projects. However, its notation did not enjoy much acceptance. The general approach of Let's Dance to first enumerate all interactions in a choreography

and then subsequently add restrictions, such as Precedence and Inhibits relationships, is similar to the modeling paradigm pursued by declarative languages, e.g. Declare [173].

While this declarative paradigm seemed attractive and suited for choreographies in the first place, the modelers involved in the projects were more used to flow-oriented modeling languages. In these kinds of languages, the set of allowed next steps is enumerated in each situation in contrast to adding constraints and subsequently restricting the allowed behavior. Especially explicit choices turned out to be hard to model in Let's Dance: the vice-versa Inhibits or two guard conditions with opposite conditions enjoyed less acceptance than explicit decision points as known from flow-oriented languages such as BPMN oder flow charts. The difference between Precedes and WeakPrecedes also seemed unintuitive to many modelers. Furthermore, the question arose why notational elements for loops and multiple instance activities needed to be reinvented although there were many notations available that already include these concepts.

These observations led to the idea of reusing as much of the popular BPMN notation as possible. All basic concepts required for a choreography language (roles, interactions, behavioral dependencies) are present in BPMN. However, a number of changes in semantics needed to be applied. This section will introduce iBPMN. It will discuss the language constructs, the formal semantics as well as the generation of BPMN observable behavior models out of iBPMN.

5.3.1 Language Overview

Elementary interactions are the basic building blocks of interaction models. While the communication activities are not reflected explicitly in iBPMN, interactions are assumed to be pairs of communication activities belonging to two roles. One activity is typically interpreted as message sending and the other as message receipt. There are a number of fundamental differences between BPMN and iBPMN:

1. *Atomicity of interactions.* While BPMN distinguishes message sending and message receipt activities, corresponding communication activity instances of two participants are assumed to happen at the same time in iBPMN.

2. *Decomposition of complex interactions.* The line-based representation of interactions in BPMN hampered the integration of a clean decomposition concept. Interaction are represented as nodes in iBPMN, allowing for graphical representation of decomposition.

3. *Assignment of control flow dependencies.* Control flow dependencies in BPMN, such as sequence flow and gateways, are assigned to the individual roles. Therefore, it is obvious who is responsible for enforcing the ordering constraint imposed by a control flow construct. In contrast to this, control flow dependencies in iBPMN are not assigned to any role. It is left unspecified who enforces a particular constraint.

4. *Ownership of decisions.* Decision gateways are also assigned to individual roles. This implies that the participant of that role is responsible for doing the choice. There is also assignment of decision ownership in iBPMN. However, the assignment is syntactically handled in a different way. One or a number of roles are specified to be responsible for that choice.

5. *Process instance creation and termination.* Process instantiation and termination is explicitly represented in BPMN through start events and end events. This indicates the lifespan of process instances. This information is not provided in the case of iBPMN. Process instantiation and termination are not defined. The latest moment of instantiation and the earliest moment of termination of a process instance can be implied.

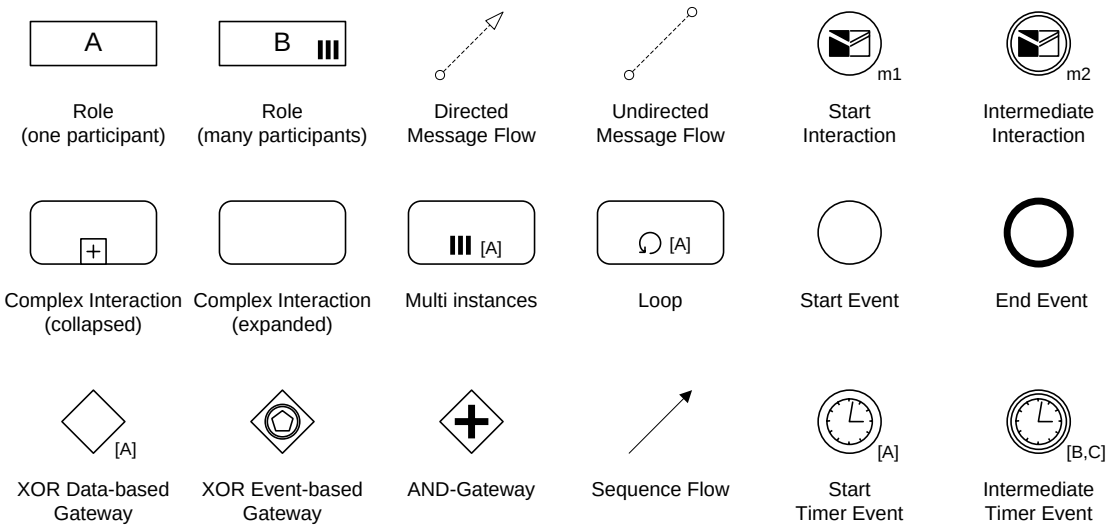


Figure 5.25: iBPMN language constructs

Figure 5.25 lists all language constructs of iBPMN. Rectangles represent roles and must not contain further elements. Similar to the extended BPMN presented in Section 4.1, we distinguish choreographies where at most one participant of a certain role is involved in a conversation from those where potentially many participants are involved. The latter case is represented using a rectangle with three bars.

Elementary interactions are represented using a letter symbol. The combination of the black and white letter alludes to message sending and message receipt in BPMN. The message type is defined for an elementary interaction using a label. Complex interactions are represented by rounded rectangles. They are either collapsed, meaning that the refinements of this interaction are hidden, or expanded, where further elements can be placed within that rectangle. Complex interactions can also be looped or defined to be multiple instances complex interactions. Again, the decision owner of how many iterations to perform or of how many instances to spawn must be defined.

A collapsed complex interaction usually references a subchoreography defined in a separate diagram. An expanded complex interaction directly contains the subchoreography. In the typical case, the same roles used in such a subchoreography are already present in the choreography containing the complex interaction. In this case, reuse of the subchoreography is a mere integration of the interaction logic into the control flow of this upper choreography. Should different roles be used, a mapping needs to be defined how the roles from the subchoreography map to the roles present in the upper choreography. Alternatively, some of them can be defined to be

additional roles.

Interactions are connected to roles via directed and undirected message flow. Directed message flow can only be used for elementary interactions, indicating who is the sender and who the receiver of a message. Undirected message flow indicates that certain roles (possibly more than two) are involved in an interaction.

Control flow branching is done through XOR data-based gateways, XOR event-based gateways and AND-gateways. In the case of XOR data-based gateways a branching condition must be present if used with multiple outgoing sequence flows. In addition, it must be specified who is the owner of the choice. This could be one role or several roles. In the latter case, the modeler must ensure that all roles are actually able to evaluate the condition. The XOR event-based gateway also indicates alternative branches. It must be followed by an elementary interaction or a timer event. Here, the branching condition is not explicitly given and also mixed choices are supported using this construct. AND-gateways can be used as splits or joins, spawning multiple threads of control or synchronizing them. Timer events also have owners assigned. Finally, start events and end events are used exactly like in BPMN.

Figures 5.26 and 5.27 illustrate the example from Section 1.1 in iBPMN. Figure 5.26 shows a high-level view of the choreography, already specifying that at most one participant of role seller and auctioning service will be involved in one conversation, while there are typically a number of bidders involved.

In addition to revealing the overall topology of the choreography, the high-level view also shows the main interaction phases in the form of complex interactions. First, the auction is set up, involving the seller and the auctioning service. Next, the auction takes place, before delivery and payment can be handled concurrently.

The high-level view is refined in Figure 5.27, illustrating how the different interaction symbols are used. A single circle represents start interactions, while the double circle represents intermediate interactions. Start interactions are entry points, where a corresponding message exchange would be the start of a conversation. Multiple start interactions can appear in one choreography, symbolizing alternative entry points into a conversation. While start interactions are the only nodes without incoming edges in a choreography, start events are the only nodes without incoming edges within any other complex interaction. A choreography is allowed to have multiple end events, while any other complex interaction must contain exactly one end event. XOR event-based gateways must only be followed by intermediate interactions or intermediate timer events.

5.3.2 Formal Semantics

Formal semantics will be provided through a translation to interaction Petri nets. However, there are a some iBPMN constructs that cannot be reflected properly in interaction Petri nets. While we distinguish directed and undirected message flow in iBPMN and even allow collapsed complex interactions, interaction Petri nets only operate on the level of elementary interactions. Therefore, the mapping will largely ignore collapsed complex interactions and requires all elementary interactions to be used with directed message flow.

Furthermore, multiple instances complex interactions cannot be properly reflected in the formal model. While spawning a (potentially infinite) number of threads can be represented in

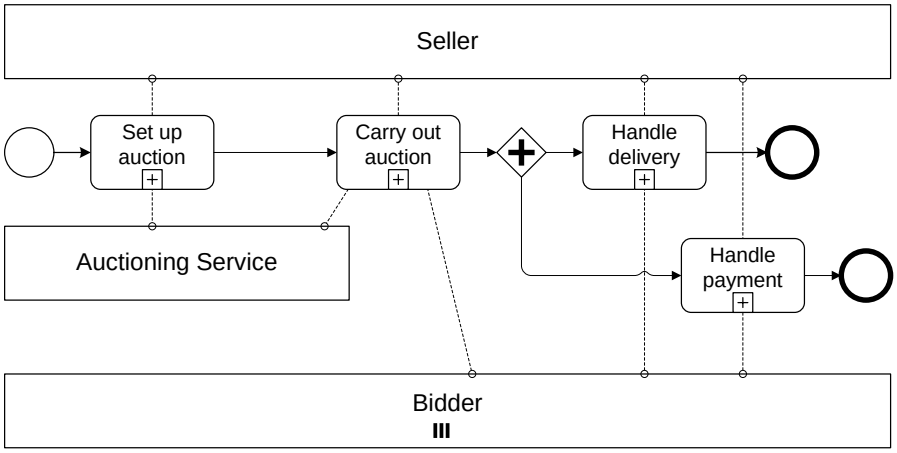


Figure 5.26: High-level iBPMN interaction model

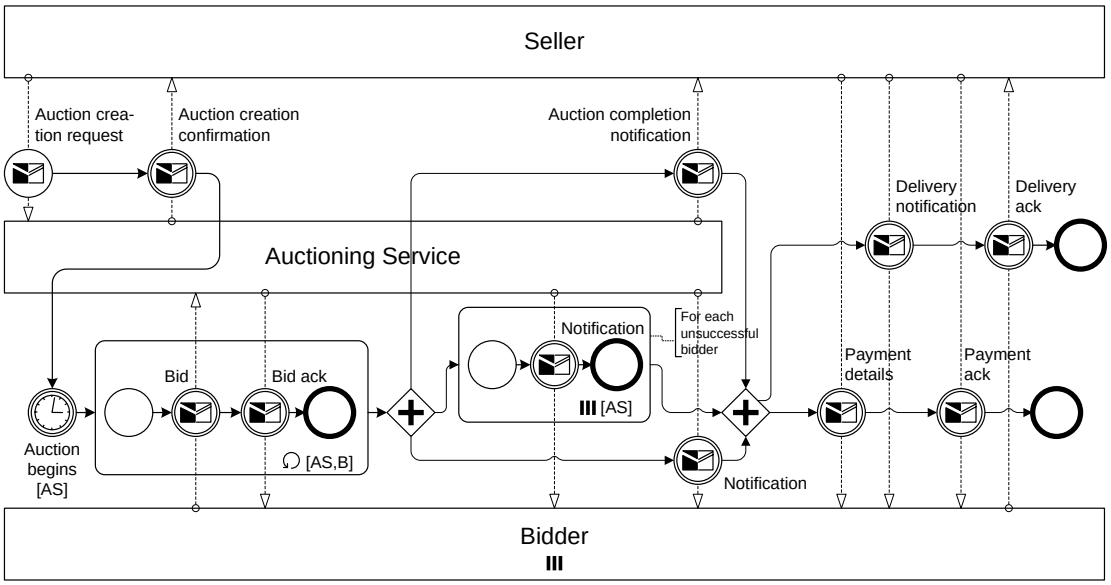


Figure 5.27: Detailed iBPMN interaction model for the bidding scenario

place / transition nets, the different threads cannot be distinguished from each other. This could lead to an undesired lack of synchronization in the presence of AND-gateways within multiple instance complex interactions. Even worse, the synchronization of different threads is only possible if the number of threads is known at design-time. The fact that place / transition nets lack support for the typical multiple instance workflow patterns [17] has already been identified by van der Aalst et al. [16]. As a workaround, multiple instance complex interactions could be mapped like looped complex interactions.

The third limitation of the mapping relates to the fact that roles with multiple participants within the same conversation are not supported by the formal model introduced in the previous subsection.

Figure 5.28 shows how the different iBPMN constructs are mapped to interaction Petri nets. For each choreography, a distinguished start place $p_{(start)}$ is created. The mapping ensures that $p_{(start)}$ is the only place without incoming flow arcs. All transitions have incoming and outgoing arcs. A place without outgoing flow arcs is created for each end event of the choreography, collectively denoted by $P_{(end)}$.

Interaction Petri nets require the definition of an initial marking and a set of final markings. As there is no corresponding concept in iBPMN, the markings have to be derived from the structure. This is easy for the initial marking. As we have a distinguished start place $p_{(start)}$ we use $[p_{(start)}]$ as initial marking. Regarding final markings, we assume that it is desirable that at most one token arrives at every end place and that no tokens remain on non-end places in any final marking. Therefore, for each non-empty subset of end places $P'_{(end)} \subseteq P_{(end)}$ we introduce a final marking m_f where $m_f(p) = 1$ for all $p \in P'_{(end)}$ and $m_f(p) = 0$ otherwise. Assuming Figure 5.1 to be the corresponding interaction Petri net for the iBPMN model from Figure 5.27, the initial marking of the net would be $[p1]$ and the final markings would be $\{[p19], [p20], [p19, p20]\}$. We see that not all valid final markings are actually reachable.

5.3.3 Validation

This section uses the requirements framework from Chapter 3 to assess iBPMN. As a graphical notation is present, requirement *C1* is supported. An iBPMN choreography can be reduced to a structural view just by omitting the control flow constructs (*C2*). Collapsed complex interactions allow to modularize a diagram (*C3*) and realize clean decomposition of choreographies at the same time (*C4*). Thanks to the possibility to map roles used in a subchoreography to the roles used in the upper choreography, iBPMN fully supports requirement *C5*.

Multi-lateral choreographies can be expressed by using more than two roles (*R1*). Ownership of choices can be specified for XOR data-based gateways, loops and multi-instance complex interactions (*R2*). Multiple participants per role are realized through a dedicated construct in iBPMN (*R3*). Time constraints can be expressed through timer events (*R6*).

Due to the absence of data flow in iBPMN, reference passing is not supported (*R4*). This issue will be discussed in the open issues presented in Chapter 7. While simple cancellation scenarios can be covered through XOR event-based gateways, cancellation of subchoreographies is not possible in iBPMN. Again, the discussion is postponed until Chapter 7.

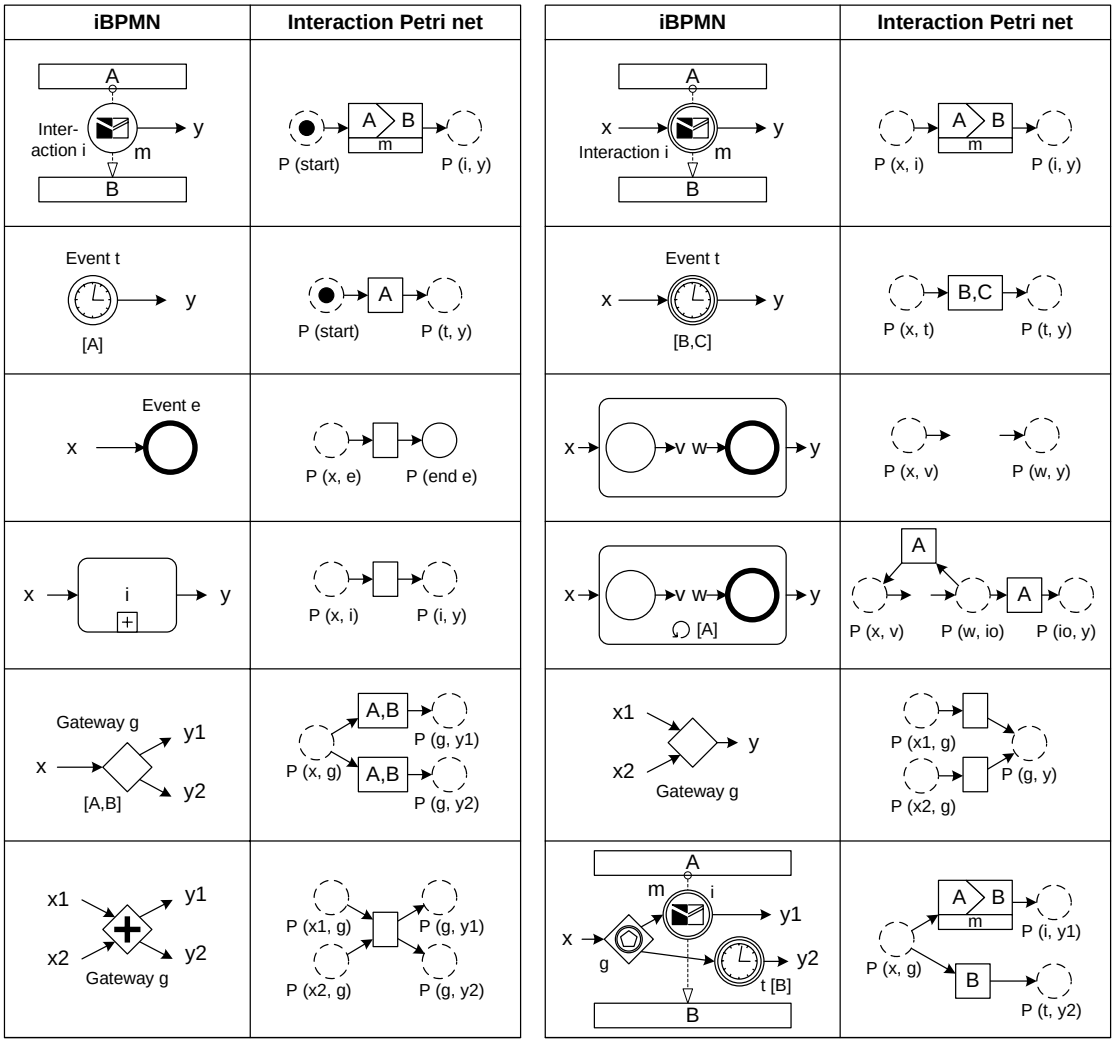


Figure 5.28: Translating iBPMN choreographies to interaction Petri nets

5.3.4 Generating Observable Behavior Models

While deriving observable behavior models for the different roles from classical BPMN choreographies is trivial, deriving these models from iBPMN choreographies is more complex. Therefore, this section discusses the challenges and presents an algorithm for deriving observable behavior models in BPMN out of iBPMN models.

A typical approach to generate observable behavior models out of interaction models is by means of model reduction (cf. [221]). Those interactions where the corresponding role is involved in are converted into corresponding communication activities. The other interactions are removed from the model while preserving the control flow dependencies.

The generation approach can be divided into three phases. (i) Interactions and other constructs are converted into communication activities and other corresponding constructs. Constructs that are not relevant for the observable behavior model of a role are converted into τ-nodes. (ii) τ-nodes are removed and control flow is rearranged accordingly. Empty complex interactions are also removed. (iii) Control flow is rearranged in a way that the syntactical constraints of BPMN are fulfilled.

(i) As illustrated in Figure 5.29(1a) through 5.29(2c), interactions are converted into corresponding communication activities or into a τ-node (depicted as crossed out rectangle). In the case of choices, the *location of choice* needs to be considered carefully. In iBPMN we have introduced ownership of choices for XOR data-based gateways, allowing to specify who is able to / responsible for evaluating the branching condition. The same concept also applies to multi instances and loops. If the observable behavior model is to be created for a role that does not own the decision, XOR event-based gateways have to be used in BPMN as the role depends on an external choice. This can easily be done for XOR data-based gateways and also for loops (which are expanded into a structure containing gateways, cf. Figure 5.29(3b) and 5.29(4b)). However, in the case of multi instances, sequentialization has to be applied due to a lack of support for the workflow pattern "multiple instances without apriori knowledge" in BPMN (cf. Figure 5.29(5b)). It is not possible to specify that parallel instances can be created on the fly while other instances are still running.

Timer events are handled in a similar way like interactions. They are either converted into a τ-node (Figure 5.29(6b)) or left as they are (Figure 5.29(6a)). XOR event-based gateways are converted to XOR data-based gateways if all following nodes are interactions and the current role is the sender. Otherwise, an observable behavior model cannot be generated. Especially mixed choices are a major challenge in this context. This issue will be discussed in more detail under the name of *desynchronizability* in Section 5.4. All other constructs are kept, including start events, end events and AND-gateways.

(ii) During the removal phase, all τ-nodes and empty complex interactions are removed. The latter also applies to those structures that were generated out of loops and multi instance complex interactions, cf. Figures 5.29(4b) and 5.29(5b).

(iii) During the rearrangement phase we mainly face the BPMN requirement that XOR event-based gateways must only be followed by (catching) intermediate events. Figure 5.30 highlights the different situations we might run into. XOR data-based gateways can be handled through duplication (Figure 5.30(1a) and 5.30(1b)). XOR event-based gateways with multiple incoming sequence flows also require duplication (Figure 5.30(2a)). AND-splits require duplication and

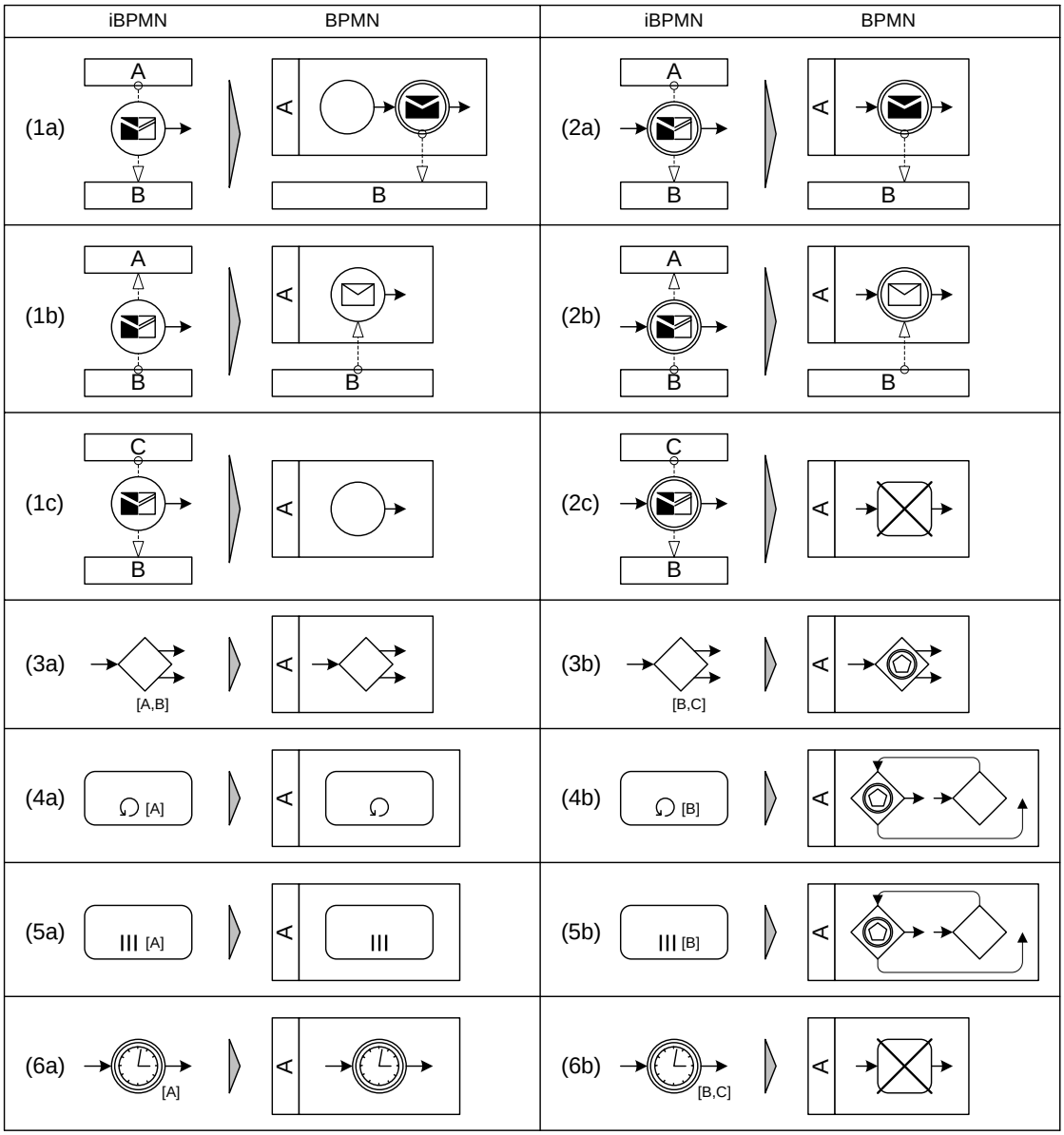


Figure 5.29: Transformation rules for the conversion phase (i)

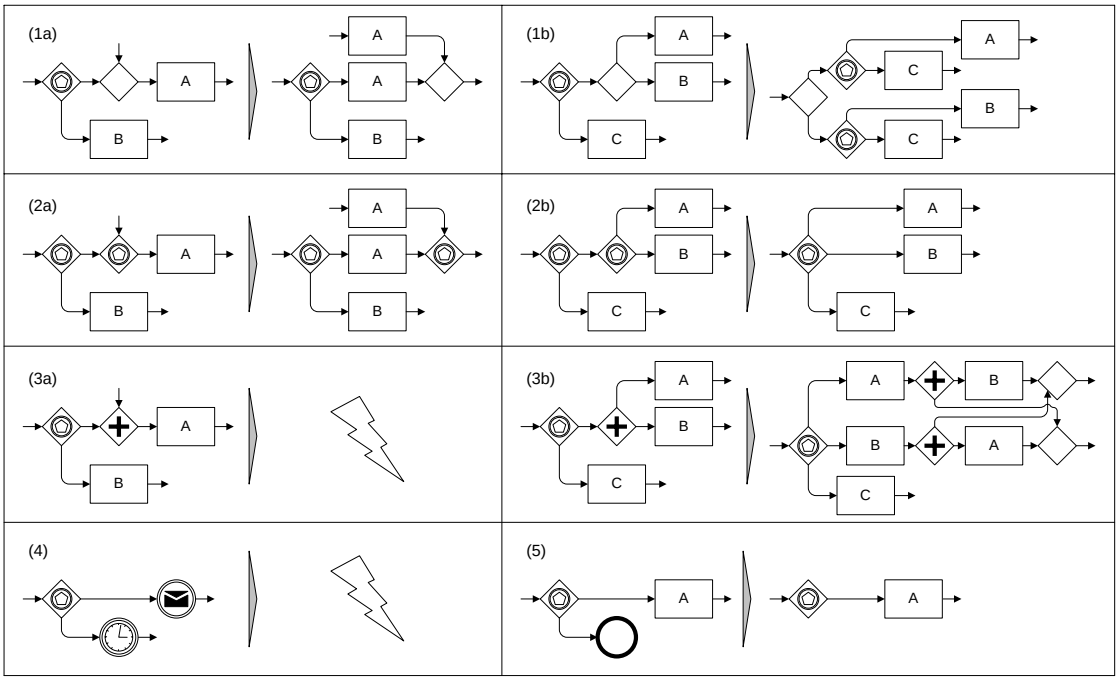


Figure 5.30: Transformation rules for the rearrangement phase (iii)

sequentialization (Figure 5.30(3b)). If a XOR event-based gateway is followed by a complex interaction, then this complex interaction needs to be expanded. In the case of a multi instance complex interaction, sequentialization needs to be applied.

We might also encounter situations that we cannot handle properly and where we cannot create a valid BPMN model. Such a situation occurs if the succeeding node is an AND-join. This situation is called *non-free-choice* in the Petri net world [92] and cannot be expressed in BPMN. It is illustrated in Figure 5.30(3a). Other problematic situations involve XOR event-based gateways followed by message send events (Figure 5.30(4)).

Figure 5.30(5) illustrates how end events are handled when directly following a XOR event-based gateway. Here, the end event is simply removed. This has the effect that the participant of that role might not be able to know whether a conversation has already ended or not.

Figure 5.31 illustrates the generated result for role seller of our iBPMN example from Figure 5.27. We see that an AND-gateway appears that has one incoming and one outgoing sequence flow. Such unnecessary gateways can be removed from the model using simple reduction techniques.

Especially the bidder role from our example reveals some of the challenges when generating observable behavior models. In the original model, a distinction is made between the bidder winning the auction and the other bidders. The winner gets the success notification and all other bidders get a notification about having lost the auction. Such a distinction between different sets of participants cannot be made in iBPMN.

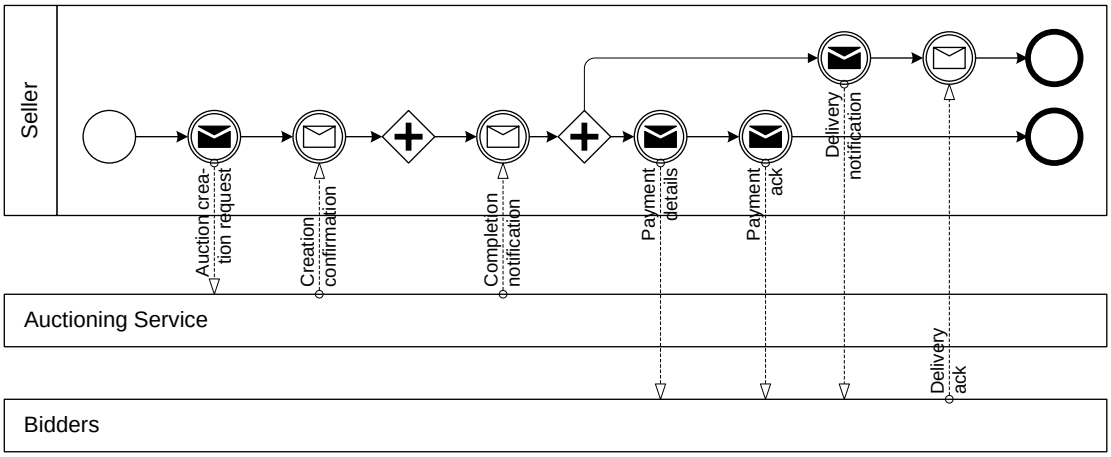


Figure 5.31: Generated observable behavior model for the seller role

5.4 Realizability

It turns out that interaction models come with a new class of anomalies. In many cases, an observable behavior model can be found for every role of an interaction model such that they collectively realize the behavior specified by the interaction model. In contrast to this, it might occur that it is impossible to find observable behavior models that collectively show this behavior. This is mainly due to the fact that the behavioral dependencies specified in the interaction model are not directly assigned to a particular role. This global nature of dependencies might lead to their unrealizability.

We can identify several issues related to the problem of realizability. The examples illustrated in Figure 5.32 through 5.35 hint to some of the subtleties.

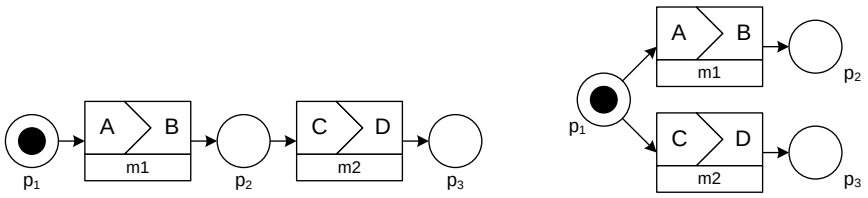


Figure 5.32: Interaction model examples

Figure 5.32(a) shows the example already described in Section 2.5.4. The final marking is $[p_3]$. It is not possible to find interacting roles that exactly show the specified behavior. The enablement relationship between the first and the second interaction cannot be realized without additional interactions as C and D cannot know whether A and B have already interacted or not. Nevertheless, it would be possible to find interacting roles that show a subset of the specified behavior: Imagine two roles A and B that interact and roles C and D simply do nothing. In this

case, however, a conversation would not terminate properly, as the final state cannot be reached.

Figure 5.32(b) shows a choice between two interactions and valid final markings $\{[p_2], [p_3]\}$. Similarly to the previous example, A and B cannot know whether C and D have already interacted and vice versa. In contrast to the previous example, we can find roles that collectively realize at least a subset of the specified behavior with proper termination. Imagine again that only roles A and B interact while C and D do nothing. In contrast to this, we are not able to find a set of roles that realize a subset of the behavior where all interactions from the conversation model are reachable.

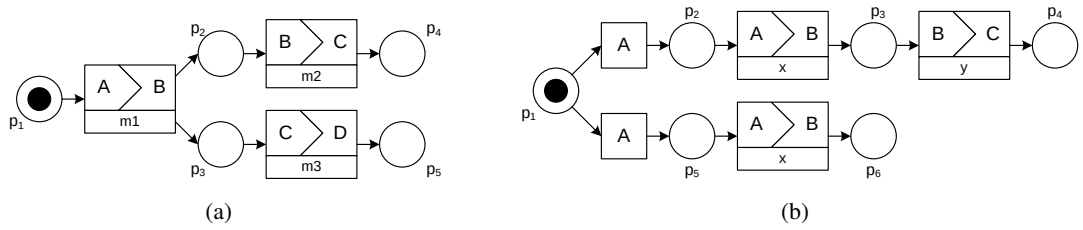


Figure 5.33: Interaction model examples

Similarly to the first example, the enablement dependency between the AB interaction and the CD interaction is the problem in Figure 5.33(a). As a solution, C could wait for the message from B before interacting with D. Once the BC interaction has happened it is clear that the AB interaction also must have happened. The resulting behavior would be a properly terminating subset of the initially specified behavior.

Figure 5.33(b) shows an example containing a non-deterministic choice. This conversation model represents that A should internally be able to decide whether B will interact with C later on. However, B cannot observe this decision as in any case it will get a message x from A. As A does not have any control over the BC interaction, the decision whether this interaction takes place or not will be independent form A's initial choice. When only considering the possible traces of the conversation model we could create roles that collectively produce exactly the same traces. The main difference is that B or C can decide whether the final interaction takes place or not in the realization. We see that considering the branching structure is crucial whenever the ownership of (and the moment of) choice is of importance.

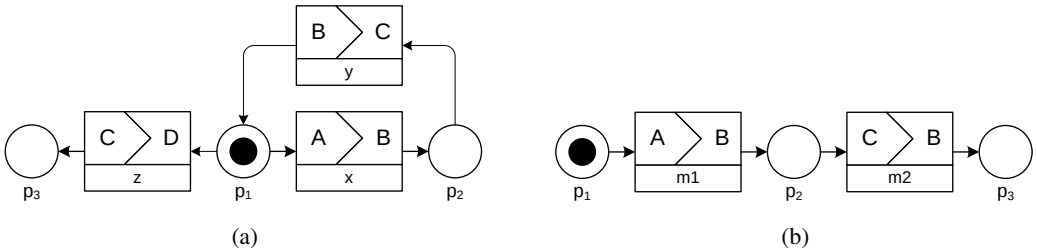


Figure 5.34: Interaction model examples

Figure 5.34(a) shows a cyclic example containing a choice between an AB interaction and

a CD interaction, similarly to the second example. The difference here is that by expanding the cycle to a sequence, we can at least find roles that realize a subset of the behavior.

Figure 5.34(b) shows an example that is perfectly realizable in a synchronous world, where C can block B until it has interacted with A. However, when considering an asynchronous world, where message sending and receiving do not happen in one step, the order of the send activities would not conform to the order of interactions in the conversation model.

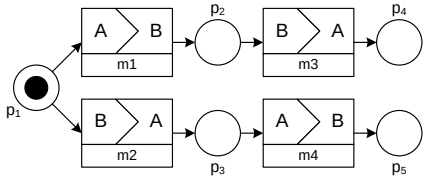


Figure 5.35: Interaction model examples

Figure 5.35 also shows an example that is realizable in a synchronous world. Here, a mixed choice between sending and receiving takes place. In an asynchronous world, it might occur that A and B send messages concurrently leading to a deadlock as the corresponding response message will not arrive.

The examples show that we need to distinguish different notions of realizability. The remainder of this section will investigate some of these notions in more detail, especially full realizability, local enforceability and desynchronizability.

5.4.1 Dimensions of Realizability

We need to consider different communication models, the complete behavior of an interaction model vs. a subset of behavior and we can consider different equivalence notions for comparing behavior.

Communication model. One option is to assume synchronous communication, where sending and receiving of messages happens at the same time. Two flavors are possible in this context: it might be allowed that a sender blocks until the receiver is ready to receive the message. Alternatively, the conversation fails if a role can only send in a given state without any other role being able to receive the message.

In asynchronous settings, message send and receive do not happen in one step. Here, message buffers are introduced for storing the incoming messages. We might assume that there is only one queue, e.g. with FIFO message delivery, or that there is a buffer where any incoming message can be received from.

The order of interactions is of central importance. However, especially in the case of asynchronous communication, there are different options of what ordering relationships to consider. For instance, only the ordering of send transitions might be considered, or the ordering of receive transitions or the ordering of communication transitions within the individual roles might be of importance.

Complete behavior vs. subset of behavior. Choreographies define constraints and obligations of the roles involved. Constraints apply as the choreography enumerates all allowed interactions in every conversation state. Obligations apply as a final state must be reached, which in turn is only possible through the execution of the given interactions.

In this context, we can either demand that it must be possible to carry out the complete behavior specified in the choreography. Or, a subset of the behavior might already be sufficient. Here, the follow-up question is what a valid subset would be. For instance, proper termination of conversations might be a basic criterion. Furthermore, reachability of all interactions from the original choreography might also be demanded.

Equivalence notion. Having agreed on what ordering relationships to consider and how much of the specified behavior is of relevance, an equivalence notion for comparing the original choreography and the collective behavior of the roles must be chosen. Here, trace-based techniques can be applied. This is sufficient when dealing with deterministic behavior in the choreography and the roles. Branching structures are of relevance in the presence of non-determinism. Here, bisimulation-like techniques can be used. There is a whole range of equivalence notions available, which are surveyed in [31, 123]. The choice of equivalence notion is highly related to the notion of conformance between an implementation and a specification.

In order to formally capture the different notions of realizability we introduce the following abstract concepts. $\oplus$ is an operator for composing observable behavior models to an interaction model. $\sim$ is a binary (and not necessarily symmetric) relation on interaction models comparing their behavior. Conversation models or interaction Petri nets can be used for describing interaction models. In contrast to this, the concepts $\oplus$ and $\sim$ remain abstract so far, being some operator and some beauty relation on choreographies. These concepts are going to be defined later on, depending on the particular notion of realizability chosen along the three dimensions. Here, $\oplus$ heavily depends on the communication model chosen and, in the case of asynchronous communication, the ordering relationships to be considered. $\sim$ depends on whether the complete or only a subset of the behavior is demanded and it also depends on the equivalence notion chosen.

Definition 5.8 (Realizability) A choreography C is realizable, iff there exist observable behavior models $P_1, \ldots, P_n$ such that $P_1 \oplus \ldots \oplus P_n \sim C$. □

5.4.2 Full Realizability

The notion of *full realizability* focuses on synchronous communication and requires the complete behavior under a bisimulation relation. Therefore, the relation $\sim$ is based on classical bisimulation, defined as follows. Regarding the operator $\oplus$ we can directly reuse the synchronous composition from Definition 5.5.

Definition 5.9 (Bisimulation) A *bisimulation* is a symmetric, binary relation R on markings of two interaction Petri nets $N_1 = (P_1, T_1, F_1, m_{01}, final_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, m_{02}, final_2, \lambda_2)$ such that $(m_{01}, m_{02}) \in R$ and for each pair $(m_1, m_2) \in R$ holds: if $m_1 \xrightarrow{t} m_1'$ then

there must exist a m_2' such that $m_2 \xrightarrow{t} m_2'$ and $(m_1', m_2') \in R$. $N_1 \sim_{bisim} N_2$ denotes that N_1 and N_2 are bisimulation-related. □

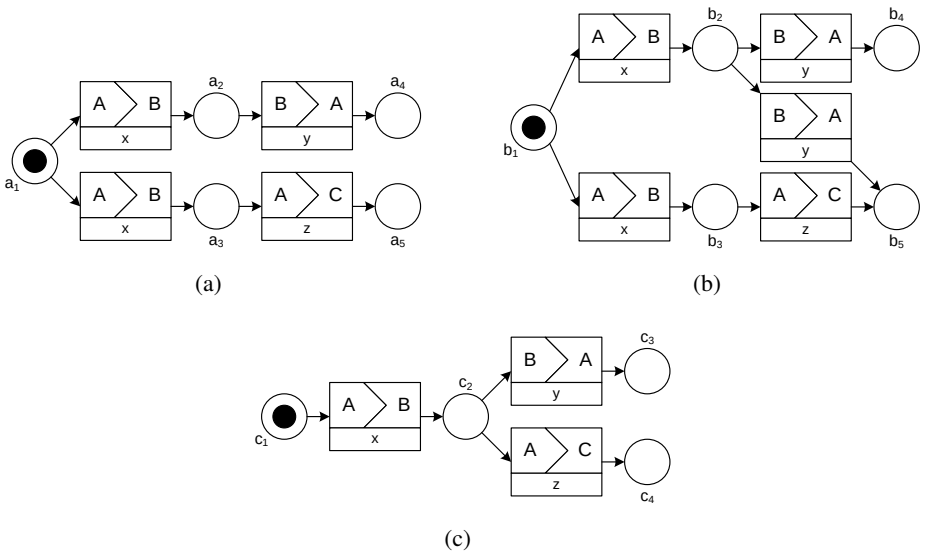


Figure 5.36: Figures 5.36(a) and 5.36(b) are bisimulation-related

Figure 5.36 shows three interaction Petri nets that can be checked for equivalence using bisimulation. The traces, i.e. the allowed sequences of interactions, of all three interaction models are identical. Yet, the decision point whether an interaction should happen between B and C or a second interaction between B and A is decided at a different point in time.

Although Figures 5.36(a) and 5.36(b) have a different structure, they are bi-simulation-related. The relation R looks as follows for this example $R = \{([a_1], [b_1]), ([a_2], [b_2]), ([a_3], [b_3]), ([a_4], [b_4]), ([a_5], [b_5]), ([a_4], [b_5]), \ldots\}$ (the other tuples are given through the symmetry of R).

Having defined bi-simulation and the synchronous join, we can now refine Definition 5.8 to *full realizability*:

Definition 5.10 (Full Realizability) An interaction Petri net C is *fully realizable* iff there exist observable behavior models $P_1, \ldots, P_n$ such that their synchronous composition is bi-simulation-related to C. □

The definition of full realizability contains an existential quantifier and the number of possible observable behavior models is infinite. Therefore, it is not straightforward to actually check full realizability. Section 5.1.3 has introduced role projection as algorithm for generating observable behavior models out of a given interaction model. The approach from [113] also resorts to role projection for realizability checking. Full realizability is slightly more challenging than the approach from [113] because bisimulation requires the branching structures to be reflected properly. Full realizability for an interaction Petri net C is given iff the synchronous composition of the role projections $\pi_{r_i}(C)$ of all involved roles r_i is bisimulation-related to the original net, i.e. $\pi_{r_1}(C) \oplus \ldots \oplus \pi_{r_n}(C) \sim_{bisim} C$.

The examples from Figures 5.34(b) and 5.35 are fully realizable.

5.4.3 Local Enforceability

Full realizability demands that the observable behavior models collectively realize the complete behavior of the interaction model. A special class of interaction models that are not fully realizable are those where at least a certain subset of the interaction model is fully realizable. Figures 5.33(a) and 5.34(a) are examples for this. In Figure 5.33(a) the second and third interactions could be sequentialized which would result in a fully realizable interaction model. In Figure 5.34(a) the sequence AB, BC, CD would be fully realizable.

The example from Figure 5.32(a) is not fully realizable, either. Removing the second interaction would also result in a fully realizable interaction model. However, we do not consider this as *valid* subset because proper termination of the choreography would be hampered. The final marking could not be reached any longer.

A subset of the interaction model from Figure 5.32(b) would be fully realizable if one of the alternatives is dropped. We do not consider this case to be a valid subset, either. We argue that every interaction appearing in the original interaction model should also be present in the subset. The argumentation behind this is that if there is no chance to reach a certain interaction in a realization, the interaction model must definitely be erroneous.

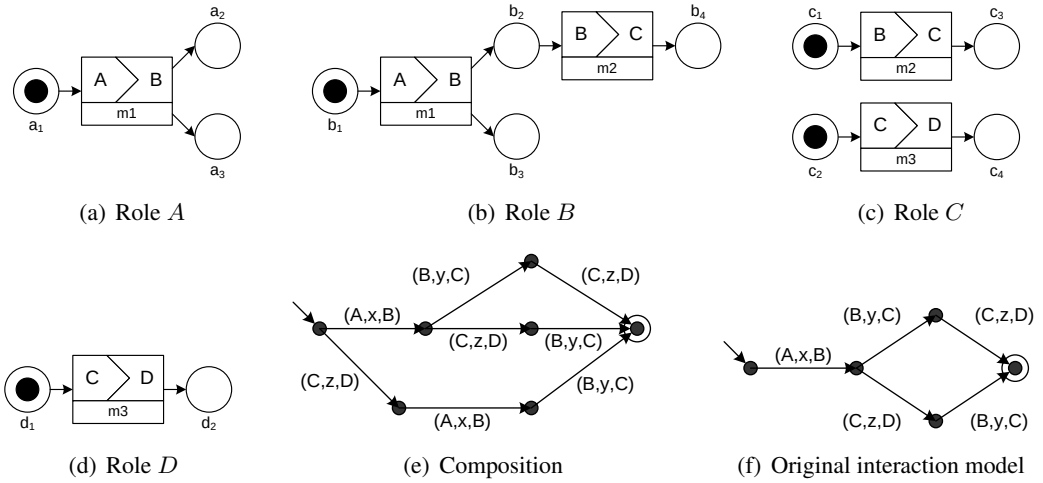
In contrast to omitting interactions in a realization, a subset of behavior (guaranteeing proper termination) might be acceptable as adding further behavioral constraints to the interaction model would lead to a fully realizable model. We call such interaction models *locally enforceable* as all constraints in the model can actually be enforced by the roles.

We resort to the notion of a *termination-preserving subset* of behavior for formally defining local enforceability. An interaction Petri net C' is a termination-preserving subset of C if there is a bijective function map relating all reachable markings in C' with a subset of the reachable markings in C, for all reachable $m \xrightarrow{t} m'$ in C' must hold $map(m) \xrightarrow{t} map(m)'$ in C and from every m where there exists a transition sequence from $map(m)$ to a final marking in C there must be a transition marking from m to a final marking in C'. Furthermore, the notion of *interaction-preservation* demands that all transitions reachable in C must also be reachable in C'.

Definition 5.11 (Local Enforceability) An interaction Petri net C is *locally enforceable* iff there exist observable behavior models $P_1, \ldots, P_n$ such that their synchronous composition is bisimulation-related to a termination-preserving and interaction-preserving subset of C. □

Figure 5.37 shows the four role projections of the interaction model from Figure 5.33(a). The reachability graph of the composition of these four interaction models is shown in Figure 5.37(e). When compared to the reachability graph of the original interaction model in Figure 5.37(f) it can be seen that the composition includes behavior not allowed by the original interaction model. The composition includes the case that interaction CD happens first. Therefore, as already mentioned earlier, the original interaction model is not fully realizable.

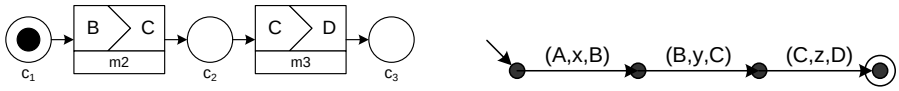
Figure 5.38(a) shows a modified observable behavior model for role C, where the two interactions have been sequentialized. When being composed with the other three observable behavior models, the reachability graph as shown in Figure 5.38(b) appears, which is a valid



(a) Role A (b) Role B (c) Role C

(d) Role D (e) Composition (f) Original interaction model

Figure 5.37: The composition of the role projections includes unallowed behavior



(a) Modified behavior for role C (b) New composition

Figure 5.38: A modified observable behavior model for C leads to a valid subset

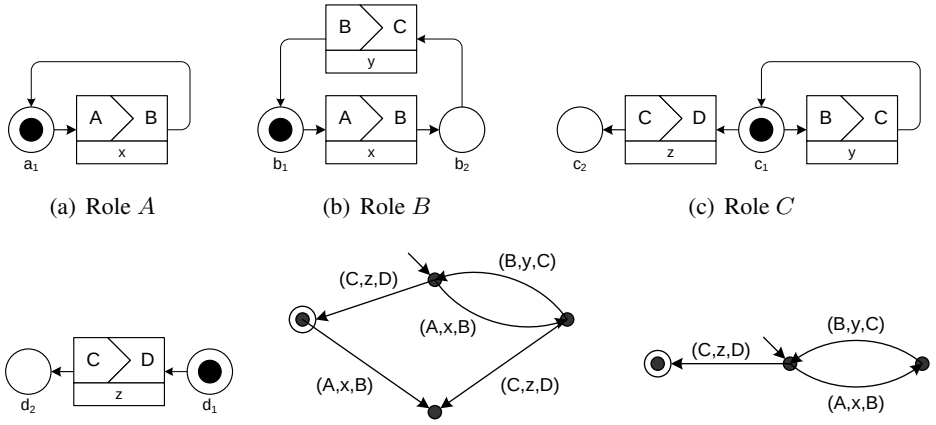
subset of the original reachability graph.

Checking local enforceability is more challenging than checking full realizability. The examples from Figure 5.37 hint to a strategy that can be applied for checking local enforceability. If it turns out that the composition of role projections is not bisimulation related to the interaction Petri net (typically because it allows more behavior), transitions from the reachability graph of the composition must be removed.

In the example, the first CD interaction must not be possible in the initial state. This can be achieved by either disallowing CD in the initial state of the observable behavior model for C or for that of role D. Disallowing it in D's model would imply that no final state can be reached (and additionally that the CD interaction will never happen). Therefore, it can only be removed from C's observable behavior model. This leads to a sequentialization of interactions BC and CD. This in turn affects the reachability graph of the composition in a second place: the CD interaction directly after AB disappears as well. The resulting reachability graph is a subset of the original reachability graph.

Each state in the reachability graph of the composition can be directly traced back to the corresponding states in the observable behavior models. In the case of interaction Petri nets, this is particularly easy: Each place in the composition model originates from exactly one observable model. Therefore, each marking of the composition can be split up to the corresponding markings of the observable behavior models.

Cycles pose a special challenge in this transition removal strategy as not only proper termination but also the reachability of all interactions must be guaranteed by the subset of behavior. Figure 5.39 shows the role projections of the interaction model from Figure 5.34(a). Again, the

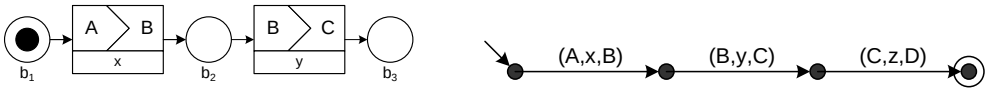


(a) Role A (b) Role B (c) Role C

(d) Role D (e) Composition (f) Original interaction model

Figure 5.39: The composition of the role projections includes unallowed behavior

composition of these observable behavior models shows more behavior than the original interaction model allows. Specifically, the interaction sequences $[AB, CD]$ and $[CD, AB]$ were not allowed in the original interaction model. According to the transition removal strategy, the AB interaction must be disallowed after the CD interaction has happened. This can be achieved by removing the AB interaction from A's or B's observable behavior model. In both cases, this would remove the AB interaction from the reachability graph altogether.



(a) Modified behavior for role B (b) New composition

Figure 5.40: A modified observable behavior model for C leads to a valid subset

Figure 5.40(a) shows an alternative observable behavior model for role B. Here, a new reachable state, namely $[b_3]$ was introduced and the iteration was removed. The composition with A's, C's and D's observable behavior models is shown in Figure 5.40(b). It shows a valid subset of the behavior of the original interaction model. Obviously, duplication of states in observable behavior models (the original marking $[b_1]$ now corresponds to both $[b_1]$ and $[b_3]$) can also lead to valid subsets.

As dealing with these state duplications would drastically increase the complexity of the strategy, only acyclic interaction models are processed in the local enforceability checking strategy. Cyclic interaction models are then covered through a preprocessing step using unfolding of the model. Here, it is sufficient to only consider those unfoldings where in any transition

sequence a transition appears at most once. Figure 5.41 shows such an unfolding of the original interaction model.

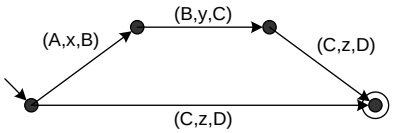


Figure 5.41: Unfolding of the acyclic interaction model

Resolving interaction models that are not fully realizable but locally enforceable is already built into the strategy presented in this section. By constructing observable behavior models that realize a subset of the behavior of the interaction model, we can derive maximal (acyclic) compositions of observable behavior models that still enforce the constraints of the interaction model. Such maximal compositions could be proposed to the modeler as resolved interaction model.

5.4.4 Desynchronizability

Full realizability and local enforceability only deal with synchronous communication. The following real-world example illustrates the shortcomings of these notions and extends on the example of Figure 5.35. This purchase order process looks as follows. A buyer submits an order to a seller. The seller returns a confirmation message to the buyer. Finally, delivery and payment are carried out.

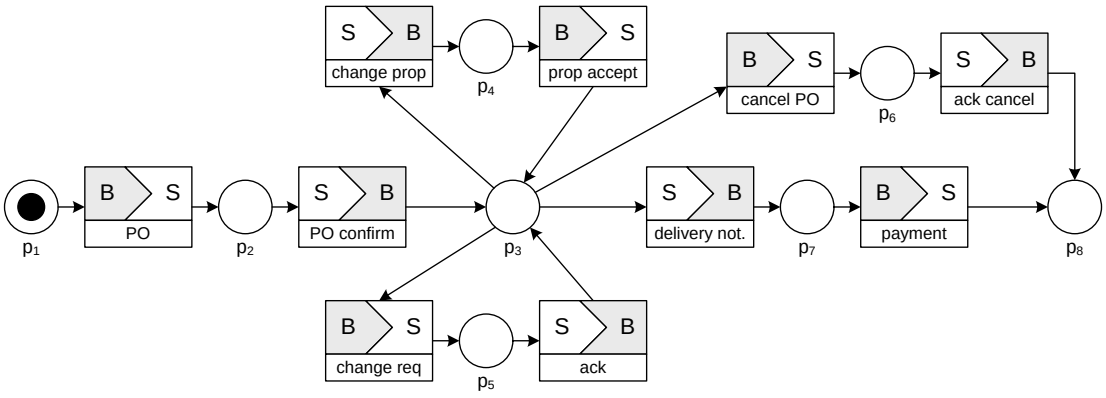


Figure 5.42: Purchase order choreography.

Several situations require a deviation from this simple process. While the seller has not sent a delivery notification and the buyer has not initiated payment yet, the buyer has the possibility to issue a change request, e. g. demanding an increased quantity. A change request must be acknowledged by the seller. The buyer might as well cancel the order which in turn needs to be confirmed by the seller. On the other hand, there might be a seller initiated change proposal,

e. g. the previous confirmation is revised by proposing a delay of the delivery date. Figure 5.42 illustrates the choreography as interaction Petri net.

The assumption that message send and receive happen in one atomic step is not valid in this scenario. The two participants might send messages concurrently. For example, the buyer's and seller's decisions to send change requests or change proposals are decoupled. Messages are sent concurrently. Therefore, we need to desynchronize the choreography to properly reflect that message sending and receiving are separate steps. Figure 5.43 shows the corresponding composed open nets.

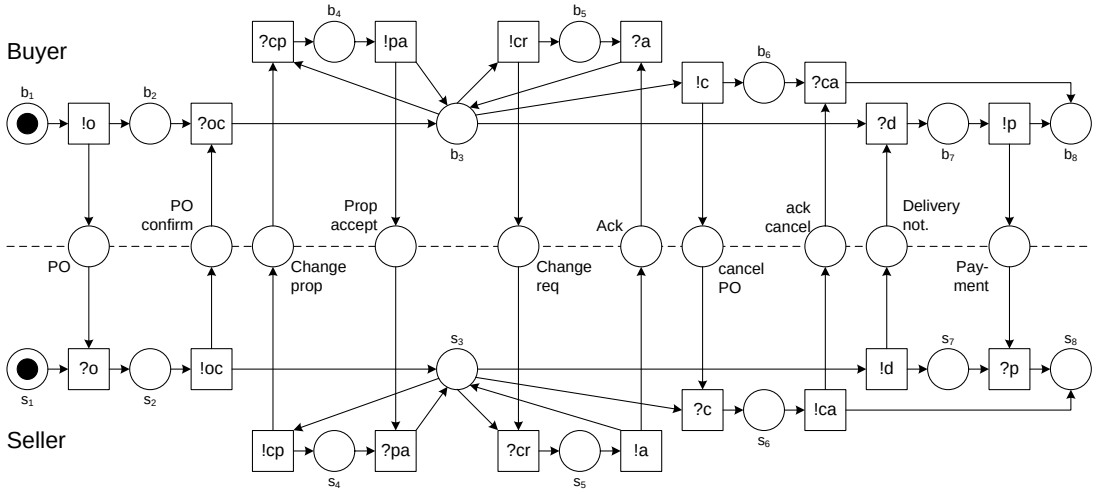


Figure 5.43: Desynchronized purchase order choreography.

The original purchase order model is fully realizable and each conversation eventually terminates in a state where there is one token left on place p_8 of the net. The desynchronized model, on the other hand, contains several problems. For instance, a deadlock occurs if the buyer decides to cancel the order and the seller proposes a change. Here, the buyer would wait infinitely for a confirmation of the cancellation and the seller waits infinitely for the response to his proposal.

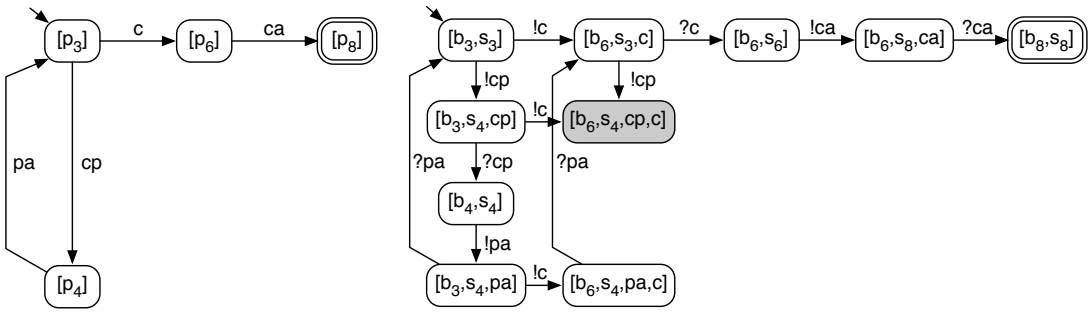
The issues presented in this example are not specific to purchase ordering scenarios. Similar issues can be found in many other areas of enterprise systems. In order to detect and locate such problems, we introduce the notion of desynchronizability.

An interaction Petri net N_s can be "desynchronized" to a net N_a using the role projections of N_s and the asynchronous composition of the resulting nets. Thereby, we introduce communication places. Whereas communication is atomic in N_s, the sending and receipt of a message is x explicitly modeled by two transitions $!x$ and $?x$ of N_a.

The desynchronized net N_a usually has more behavior than the original interaction Petri net N_s: The atomic message transfer in N_s can be mimicked by N_a by firing first the sending and then the receiving transition. Moreover, it might also be possible that N_a can fire a transition in an intermediate state introduced by the decoupling of sender and receiver. If this additional

behavior does not jeopardize weak termination, N_s is *desynchronizable*.

Definition 5.12 (Desynchronizability) Let N_s be a weakly terminating and fully realizable interaction Petri net. N_s is *desynchronizable* iff the desynchronized net N_a for N_s weakly terminates. □



(a) synchronous choreography (b) asynchronous choreography with deadlock

Figure 5.44: Reachability graphs showing the example's race problem.

The most frequent reason for non-desynchronizable choreographies are mixed choices, where there is a conflict between transitions with different senders. As mentioned earlier, the desynchronized example choreography (see Fig. 5.43) contains a deadlock. Thus, the original choreography in Fig. 5.42 is not desynchronizable. This can be detected by analyzing the reachability graphs of the nets. In Fig. 5.44(a), a part of the reachability graph of the original choreography is depicted. In the marking $[p_3]$ the transitions c and cp are (among others) enabled. The same situation is depicted in Fig. 5.44(b) for the desynchronized net. Here, the transitions $!c$ and $!cp$ are enabled in $[b_3, s_3]$, but can occur concurrently: neither transition disables the other, and a deadlocking marking $[b_6, s_4, cp, c]$ is reachable when firing these transitions in either order.

Definition 5.13 (Conflict Transitions) Let N_s be a non-desynchronizable interaction Petri net and N_a the desynchronized net for N_s. Define the set of *conflict transitions* T_C to contain all transitions t of N_a such that: there exists a marking m with $m \xrightarrow{*} m_f$ for a marking $m_f \in \mathit{final}$, and there exists a marking m' with $m \xrightarrow{t} m'$ and $m' \not\xrightarrow{*} m'_f$ for any $m'_f \in \mathit{final}$. □

The set T_C contains all transitions whose firings can make a final marking unreachable. From Fig. 5.44(b) we can conclude that the transitions $!c$ and $!cp$ are conflict transitions for the desynchronized net of Fig. 5.43. With state-of-the-art Petri net model checkers such as LoLA [193], conflict transitions can be detected efficiently even for larger choreographies.

While mixed choices are a typical reason for race problems, not all mixed choices are problematic (see Fig. 5.45). Here, both roles can send a message before receiving one. However, due to the follow-up interactions, the desynchronized choreography weakly terminates.

Finally, conflict transitions do not necessarily need to be in a structural conflict, i. e. sharing common input places. Figure 5.46(a) shows a synchronous choreography that weakly terminates with final marking $[p5]$. Here, the choice whether the transition regarding message v or the transition regarding w fires first influences what transitions will be enabled later on. If the

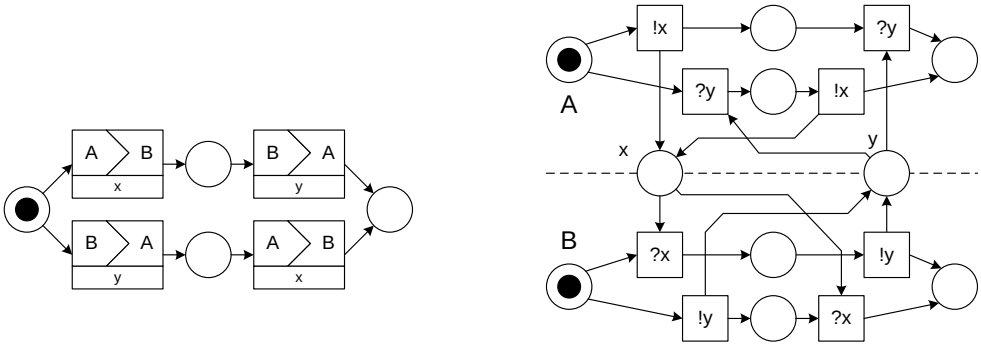
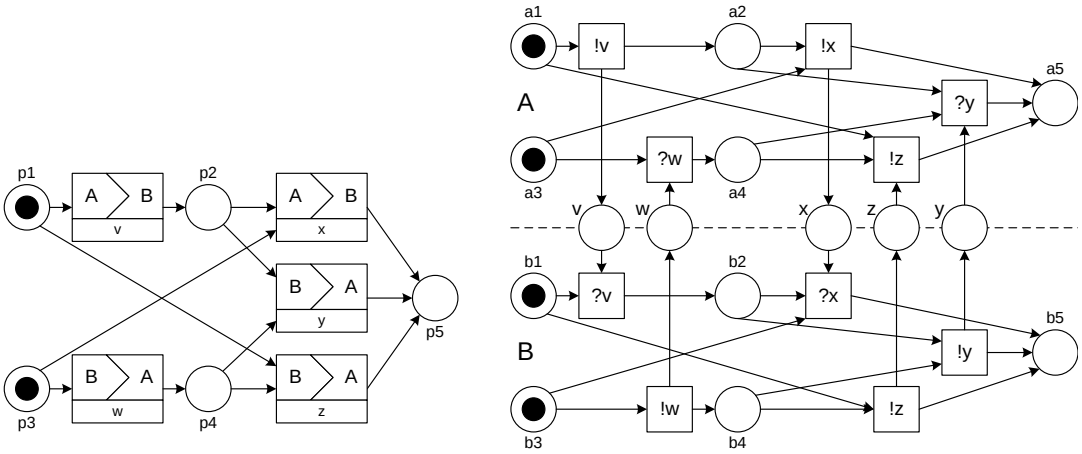


Figure 5.45: Structural conflicts do not necessarily lead to deadlocks.



(a) synchronous choreography

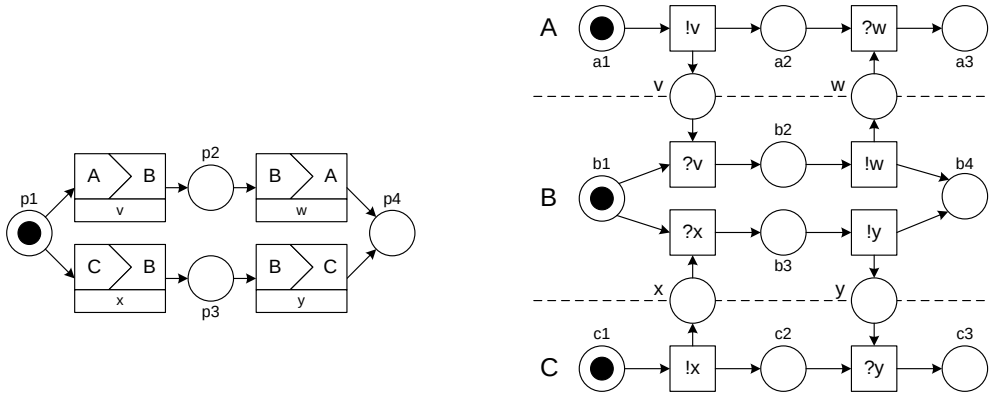
(b) desynchronized choreography

Figure 5.46: Conflict transitions without structural conflict.

transition involving v fires, the transition involving z will not be enabled any longer.

The definition of conflict transitions also captures those scenarios where individual roles are able to send messages in a final marking. Figure 5.47(a) shows an example involving three roles and final marking $[p4]$. Figure 5.47(b) shows the desynchronized choreography as generated by the algorithm in [89]. The final markings for the individual role projections are $[a_1]$ and $[a_3]$ for A, $[b_4]$ for B and $[c_1]$ and $[c_3]$ for C. This results in the valid final markings $[a_1, b_4, c_3]$, $[a_3, b_4, c_1]$, $[a_1, b_4, c_1]$, and $[a_3, b_4, c_3]$ for the desynchronized choreography, while only the first two markings are actually reachable. In marking $[a_1, b_4, c_3]$ role A is ready to fire transition $!v$. Firing this transition actually leads to a marking from where no valid final marking can be reached any more. Therefore, $!v$ is a conflict transition.

Provided that the original interaction Petri net weakly terminates there will at least be one firing sequence leading to a final state. Therefore, conflict transitions can always be found if the choreography is not desynchronizable. This guarantees that the modeling errors can be located in the original interaction model. This in turn is important for the resolution of the issue.



(a) synchronous choreography (b) desynchronized choreography

Figure 5.47: Deadlocks caused by firing transitions in a valid final marking.

5.4.5 Resolution Strategies for Non-Desynchronizability

The definitions from the previous section enable the identification of conflict transitions. Therefore, modeling errors can already be located and the modeler is made aware of the fact that the model needs to be modified or extended. As a next step, a strategy has to be chosen to remove desynchronizability problems from a choreography. Several strategies are applied in real-world implementations. Each strategy comes with a set of implications on the business level that need to be carefully considered before applying them. The strategies presented in this section have been distilled from implementation projects at SAP.

In the remainder of this section we will use the term *conflicting messages* for a pair of messages sent by different partners that correspond to a pair of conflict transitions as defined in the previous section. Both messages must belong to the same conversation.

It could be a possible strategy to resolve the choreography in such a way that conflicting messages simply cannot occur any longer. For the example, this would mean that there is no chance of having a delivery notification and a cancel message been sent in the same conversation. This could be achieved e.g. through total sequentialization of the choreography, where only one partner is allowed to send messages at a time. However, such a strategy is typically not feasible in real-world scenarios. It is often desired that conflicting messages are possible—but for the case that this occurs, a predefined resolution must be in place. Therefore, we are going to list different strategies applied in practice that follow this approach.

The following strategies can be categorized into two groups. Either (a) there is a predefined outcome upon conflicting messages, most typically one message is considered and the others are ignored, or there might be different outcomes possible. Here, we can again distinguish three types: (b) one partner could be allowed to determine the outcome and tell the other partners his decision; it could also be defined that (c) each partner decides individually for the outcome, or that (d) there is a negotiation regarding the desired outcome.

Precedence

The general idea is to define precedence relationships at design-time, prescribing how partners have to behave in the case of conflicting messages. Therefore, precedence mostly falls into category (a). If a partner detects conflicting messages, he knows the outcome of this conflict and can immediately continue accordingly. He assumes that the other partners will also detect this conflict sooner or later and also act accordingly.

The definition of precedence relationships must not be seen as pure technicality as it has direct business impact. Therefore, precedence relationships would need to be part of interaction contracts. Regarding the definition of precedence relationships we distinguish three different strategies.

Singular Interaction Partner Precedence. This strategy looks at individual interactions, e. g. the cancellation interaction in our example, and defines precedence of one partner over the other. Here, we can distinguish between two settings: (i) the buyer has precedence over the seller or (ii) the seller has precedence over the buyer.

Case (i) means that if the buyer sends the cancellation, the seller has to accept the buyer's cancellation in any case. This means that the buyer can assume that the cancellation message will have the desired effect, once it has been sent. Therefore, the seller does not need to return any confirmation message in this case. This corresponds to category (a).

In case (ii) the seller has a veto right regarding cancellation messages sent by the buyer. The seller can accept this request and return a cancellation confirmation. Only now the buyer can be sure that cancellation was successful. As an alternative, the seller could also send a cancellation rejection. Therefore, the seller can decide on the outcome, implying category (b).

Deciding for each interaction for a partner precedence individually does not solve race problems in the general case. If, for example, we decide that the buyer has precedence regarding buyer initiated cancellation and the seller has precedence regarding seller initiated change proposals, deadlocks are still possible. Now imagine the opposite setting where the seller has precedence regarding buyer initiated cancellation and the buyer has precedence regarding seller initiated change proposals. Here, the partners have veto rights for the corresponding requests. If the buyer sends a cancellation request and the seller sends a change proposal at the same time, the buyer will reject the change proposal as it conflicts with the previously sent cancellation request. The same holds true for the seller reacting to the cancellation request. After both partners have rejected the respective requests, they can, of course, resend their requests.

Type-based Precedence between Multiple Interactions. While the previous strategy considered interactions individually, this strategy considers precedence regarding combinations of interactions. Here, the message types are considered and always a fixed outcome is defined, therefore category (a). A crucial aspect of this strategy is that no combination of interactions is forgotten.

A precedence rule could be that a delivery notification has precedence over buyer initiated cancellation messages and buyer initiated change requests. On the other hand, a buyer initiated request always has precedence over seller initiated change proposals. Figure 5.48 illustrates a

resolved desynchronized choreography for this precedence rule. This resolved version weakly terminates. All transitions that were added to the original Petri net have a striped background.

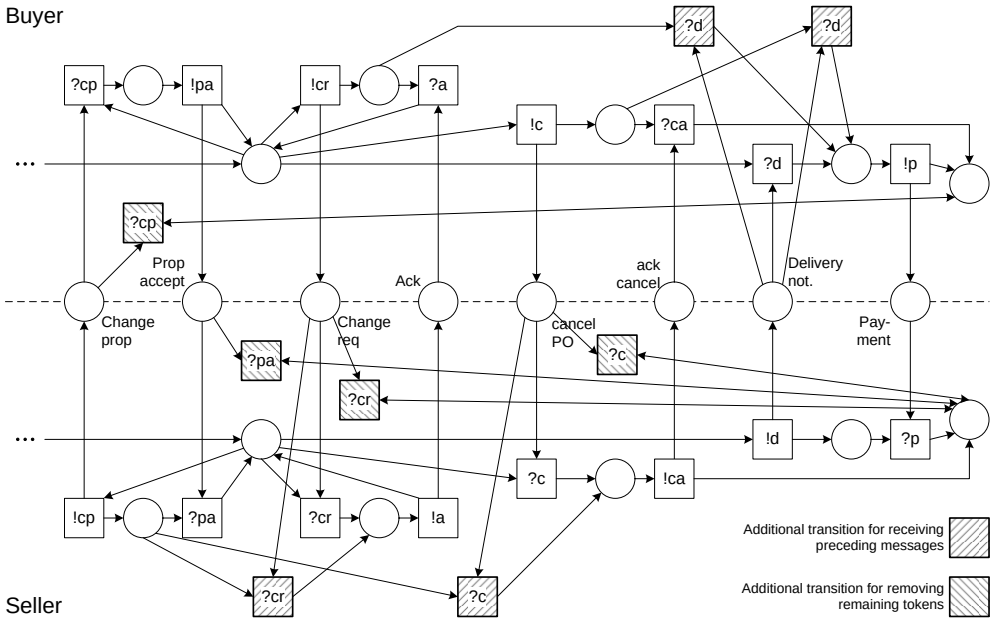


Figure 5.48: Resolved purchase order choreography.

A conflict between a seller initiated change proposal and a buyer initiated change request is resolved in the following way. In addition to being ready to consume an acceptance message for the change proposal, the seller can also consume a change request or a cancel message instead. This is manifested through the additional transitions $?cr$ and $?c$. The change proposal message must finally be consumed by the buyer without having any effect on the buyer. This happens through the additional $?cp$ transition.

The proposed solution in Fig. 5.48 is still not optimal from a business point of view. If the seller sends a change proposal while the buyer sends a change request, the messages conflict and the seller will receive the change request and accept it. In this situation, the seller assumes that the buyer will ignore the change proposal. However, the buyer could receive the accept message first and receive the change proposal afterwards. Now, the buyer cannot know that this change proposal conflicted with the change request and therefore accepts the proposal. However, the seller is not able to receive this message and will only remove the remaining token at the end of the choreography. From a business point of view this behavior is undesired: the buyer has accepted a change proposal that the seller assumes obsolete.

Another problem might arise when precedences are cyclic: Imagine there are three partners A, B and C. A can send a message to B (interaction AB), B to C (BC), and C to A (CA). BC has precedence over AB, CA has precedence over BC, and AB has precedence over CA. Now a conflict involving all three interactions occurs. Every partner thinks that his message has precedence over the message received and simply ignores the incoming message. This again could result in a deadlock.

Situation-based Precedence between Multiple Interactions. While precedence rules between different interactions were based on message types, this strategy allows more fine-grained precedence rules and again falls into category (a).

Imagine a logistics scenario where a customer lets a shipper transport his goods. The shipper selects different carriers and creates a shipment that he sends to the customer and which needs to be commented by the customer. At the same time, the customer has the possibility to cancel his order. While cancellation precedes the shipment plan interaction in the default case, this is only true for the first two weeks after the initial order. After these two weeks have passed, the shipment plan interaction precedes the cancellation. This might be due to the fact that cancellation at this point in time would simply involve too much cost. However, while the shipment plan has not been finalized yet, the customer can still cancel the order.

An underlying assumption of this strategy is that both partners come to the same conclusion about precedence. As time is the criterion in this example, both need a common understanding about when the two weeks have passed. Therefore, the arrival time of the message cannot be used as criterion, as the corresponding sender might not be able to know when this is.

Allowing Individual Decisions

Allowing individual decisions leaves it open to every partner involved to decide for an outcome individually: category (c). In the case of a buyer initiated cancellation request conflicting with a seller initiated change request, there are two alternatives for each partner:

- The seller either (S1) rejects the cancellation request and assumes that the change request has still relevance or (S2) accepts the cancellation request and assumes that the change request is obsolete.

- The buyer either (B1) rejects the change request and assumes that the cancellation request has still relevance or (B2) accepts the change request and assumes that the cancellation request is obsolete.

Out of these possibilities two are ideal outcomes: The combinations (S1)+(B2) and (S2)+(B1) lead to the acceptance of exactly one request. Even the combination (S1)+(B1) is acceptable, as the conversation is exactly in the same state as before the two requests and requests can be issued again. Maybe this time, one of the partner succeeds with his intent.

Only the combination (S2)+(B2) is problematic as both requests were accepted and both partners assume a wrong situation. However, once an accept message arrives, the conflict is detected and a resolution can be achieved as described in the other strategies.

Although this strategy does not guarantee a proper resolution in the general case and requires resorting to other resolution strategies, it is still worth considering as most outcomes are acceptable. A major challenge of this approach is that the process instances need to be realigned in case a partner has already continued, assuming his decision led to an acceptable situation. This might involve compensation and becomes especially difficult if communication to other partners is involved.

Negotiation of Outcome

Negotiation is another strategy where the outcome of conflicting messages is not fixed, therefore category (d). Here, the different partners need to reach agreement about the outcome. Such negotiation can either happen through human intervention or automatically. Human intervention could simply involve a phone call or an email exchange. In many cases such human intervention is actually desirable. For instance, the cost of cancellation might increase depending on what actions the partner has already performed. Therefore, it could be negotiated whether cancellation is still desired under the new conditions.

As an alternative, a formal hand-shake to support negotiation could be factored into each partner's process. For this, all partners need to agree on conflicting messages requiring negotiation and implement common exception handling logic. This would involve strictly sequential interactions, as partners arbitrarily reciprocate to resolve the conflict. First, conflicting messages would be detected by a partner and be broadcast to relevant partners. Each partner's process would be required to escalate to its common exception handling logic such that all parts of the process impacted by the conflict are suspended. The first part of the exception handling logic would be to determine which partner gets the write token. This remains an open issue although some basic heuristics could be defined, e. g. the first partner detecting the conflicting messages gets the write token. Another serious issue is managing multiple conflicts which can arise concurrently and determining the priority in which they should be handled. These and other issues have been handled in techniques applied for self-stabilizing systems [97].

5.5 Discussion and Summary

This chapter has introduced interaction Petri nets as formal model for interaction models based on labeled Petri nets. One of the central differences between interaction Petri nets and conversation models is that interaction Petri nets support the notion of concurrency. It might be argued that the notion of concurrency does not necessarily need to appear in choreography formalisms. Plotkin and Pratt even prove that concurrency, more specifically partially ordered multisets, are unobservable when assuming that the observers are isolated individuals [177].

All choreography languages that have been proposed (cf. Section 2.3) include the notion of concurrency. The introduction of concurrency also into the formal model eases the mapping between choreography language and formal model. Therefore, interaction Petri nets proved valuable for specifying the formal semantics for Let's Dance and iBPMN. However, due to its limited expressiveness, some of the constructs could not be mapped properly, e.g. the for each construct in Let's Dance and the multi instance subchoreographies in iBPMN.

The investigations on realizability in Section 5.4 are independent of particular choreography languages. The notions full realizability, local enforceability and desynchronizability apply to Let's Dance, iBPMN, as well as to WS-CDL, UML Communication Diagrams and BPSS.

Let's Dance and iBPMN have been introduced as novel interaction modeling languages. Both languages have similar suitability for most recurring scenarios (from the perspective of how many modeling elements are needed to represent the scenario). In contrast to this, first practical insights into these two languages show a significant difference in the acceptance by human

modelers. While the training effort needed to teach the two modeling languages to modelers is similar, the correct usage of the modeling elements differed dramatically in the modeling projects. The three different edge types in Let's Dance confused the modelers and Precedes and WeakPrecedes relationships were seldomly used correctly from the start. Especially the correct representation of optional interactions in a choreography, which requires the use of guarded interactions in combination with WeakPrecedes relationships, turned out to be a major challenge. Or, simply the fact that alternative branches are modeled using a guard condition and the inverted guard condition, rather than modeling an explicit decision point leading to one or the other branch, seemed counter-intuitive. Inhibits relationships were hardly used.

iBPMN, in contrast, enjoyed a broad acceptance by the modelers right from the start. Borrowing most of its notation from the Business Process Modeling Notation turned out to bring much of BPMN's popularity also to iBPMN. iBPMN also performed well in comparison with the interconnection modeling style in terms of popularity. When being given the choice whether to use Let's Dance, BPMN or iBPMN for choreography modeling exercises, undergraduate students following the BPM lectures at the Hasso-Plattner-Institute chose iBPMN in 86% of the cases and BPMN in 14% of the cases. Let's Dance was never picked as favorite choice.

The observation of anti-patterns in modeling practice was one of the main motivations behind investigating the interaction modeling style. At this point we can evaluate which of these anti-patterns can actually be avoided using a language such as iBPMN.

Incompatible branching behavior (*D1*) stems from the fact that the ownership of choice is not properly modeled. In contrast to this, iBPMN enables the modeler to first reflect that alternatives exist at a certain point in a choreography. Only in a second step, the modeler needs to think about who actually owns the choice or who is able to make the choice. That way, incompatible branching behavior can be avoided in most cases.

In the case of impossible data-based decisions (*D2*) the decision owner actually does not have the possibility to make the decision. This problem cannot be avoided using iBPMN or other interaction modeling languages.

Mixed choices (*D3*) can be represented in iBPMN and Let's Dance. iBPMN offers event-based gateways for this purpose and Let's Dance offers vice-versa Inhibits relationships.

Contradicting control flow (*O1*) cannot occur in interaction models as the control flow is defined on a global level which avoids redundancy and contradictions. Also the risk of incomplete control flow (*O2*), a common mistake in interconnection models, can largely be avoided in interaction models. Uni-lateral sequentialization (*O3*) is not possible in interaction models, as the control flow is defined globally. Restricting control flow constraints for individual roles only, as it must be present for this anti-pattern, is not possible.

Optional participation (*I1*) and not-guaranteed termination (*I2*) are not an issue in interaction models. As process instantiation and process instance termination are out of scope, corresponding issues only arise on a lower level.

We can conclude that we do not face many of the anti-patterns in interaction models (*D1*, *D3*, *O1*, *O2*, *O3*). Two anti-patterns are out of scope of interaction models (*I1*, *I2*). Only one anti-pattern, namely *D2*, remains problematic also when using the interaction modeling style.

Chapter 6

Implementation

Tooling plays an increasingly important role in the academic business process management community [204, 186]. There are good reasons for this fact. Firstly, theoretical concepts benefit from exploration using prototypical implementation of the concepts. By experimentation based on real-world business processes, concepts can be evaluated and refined. Secondly, the practical applicability of the research work can be demonstrated, which is important to raise awareness of academic BPM research to practitioners.

In academic research groups, researchers tend to implement small-scale prototypes that can do exactly what the particular researcher is interested in. Typically each project is started from scratch. If results from collaborators are re-used, then re-use is done in a non-structured way, by copying and pasting program code. As a result, the wheel is re-invented many times, and valuable resources are wasted. Motivated by this observation, the business process technology research group at HPI has decided to develop an open and extensible framework for business process management, called Oryx[1].

Oryx supports web-based modeling of business processes using standard web browsers, so that no additional software installation at the client side is required. Users log on to the Oryx web site and authenticate by openID[2], an Internet authentication standard. They start modeling processes, share them with collaborators, or make them available to the public.

More technically, each model artifact is identified by a URL, so that models can be shared by passing references, rather than by exchanging model documents in email attachments. Since models are created using a browser and models are just "a bookmark away", contribution and sharing of process models is eased. Using a plugin mechanism and stencil technology, Oryx is extensible. Today there are stencil sets for different process modeling languages, including BPMN, and Event-driven Process Chains (EPC [137]), Petri nets, iBPMN, Let's Dance and others. But the extensibility is not restricted to process languages. The plugin mechanism also facilitates the development of new functionality, for instance mappings to executable languages, thereby providing a business process management framework.

Four choreography language proposals and a set of verification techniques have been pro-

[1] See `http://oryx-project.org`
[2] See `http://openid.net/`

posed in the course of this thesis. A variety of extensions to Oryx have been implemented for these approaches. The following section presents the architecture and extension mechanisms of Oryx. The other sections introduce the implementations for the different languages and approaches presented in this thesis, namely for BPMN, BPEL4Chor, instance isolation checking, Let's Dance, iBPMN and realizability checking.

Oryx Architecture and Extension Mechanisms

Oryx is built to support the following three main use cases. (1) Most importantly, there must be editing functionality for graphical business process models. Different modeling languages are present in the BPM field. Most prominently, there are the Business Process Modeling Notation (BPMN [6]) and event-driven process chains (EPC [137]). Here, the corresponding stencil sets must be available and language-specific constraints on the models must be enforced. E.g. it must not be possible to connect two events in an EPC or to have incoming sequence flow into a start event in BPMN.

(2) Once process models have been created, it must be verified if the models are free of modeling errors. As a precondition for such analysis, elements must be properly connected, for instance BPMN tasks with preceding or succeeding tasks, or tasks with their parent subprocess or pool. BPMN comes with a special challenge, namely attached intermediate events, where a node is directly connected to another node without edges in between.

(3) Process models are subject to transformations. E.g. BPMN models must be transformed to Petri nets in order to carry out analysis. In other scenarios, high-level models serve as input for generating stubs for more technical models. As an example, BPMN models can be transformed to BPEL processes. In order to ease integration with other systems, common interchange formats must be supported.

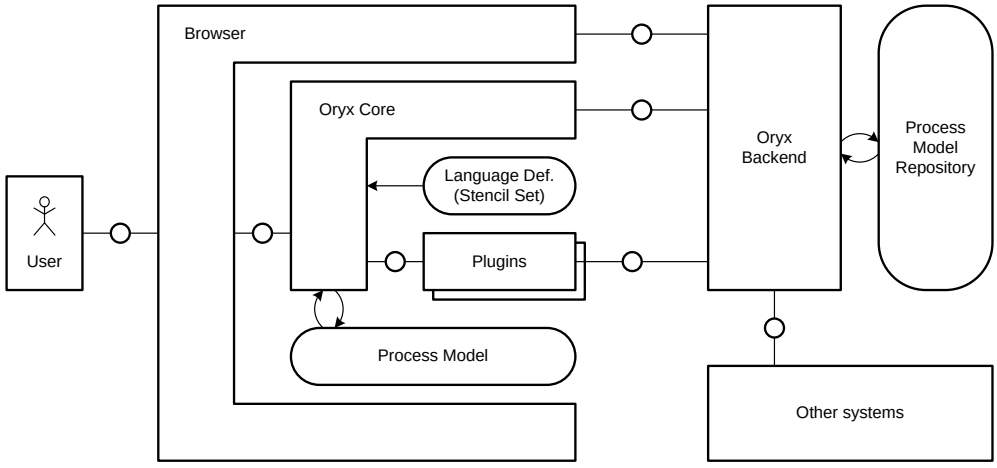


Figure 6.1: Oryx architecture

Figure 6.1 depicts the Oryx architecture. Oryx Core is a set of JavaScript routines loaded into a web browser as part of a single document describing the whole model. Models are represented

in RDF format. Oryx Core provides generic handling of nodes and edges; how to create, read, and update them; as well as an infrastructure for stencil sets and plugins.

1. *Language Support via Stencil Sets.* Stencil sets drive the Oryx Core, as they provide explicit typing, connection rules, visual appearance, and other features differentiating a model editor from a generic drawing tool. Oryx today has full support for BPMN 1.2. In addition, there is a stencil set for EPC and Petri nets.

2. *Feature Extensions via Plugins.* Plugins allow for both generic as well as notation-specific extensions. E.g. element selection and cut & paste are plugin features, as they are not needed for an Oryx viewer. More advanced plugins allow for complex model checking beyond the powers of the stencil set language.

3. *Data Portability beyond Oryx.* The Oryx Core, with the help of stencil sets and plugins, allows users to create, edit, and view visual models within a browser. Currently, Oryx does so by self-modifying the loaded page and sending it back to the server in whole. Being web-based, Oryx reduces deployment and collaboration to distributing a single bookmark.

More details on Oryx can be found in [81, 82, 145, 71, 67].

BPMN Implementation

BPMN is the main process modeling language supported by Oryx. Graphical editing capabilities are available for this language and BPMN shapes can be composed to a BPMN diagram. Connection rules ensure that the syntactical constraints of BPMN are ensured. On top of this basic functionality, called BPMN editor in Figure 6.2, a number of additional software components have been implemented in the context of this thesis.

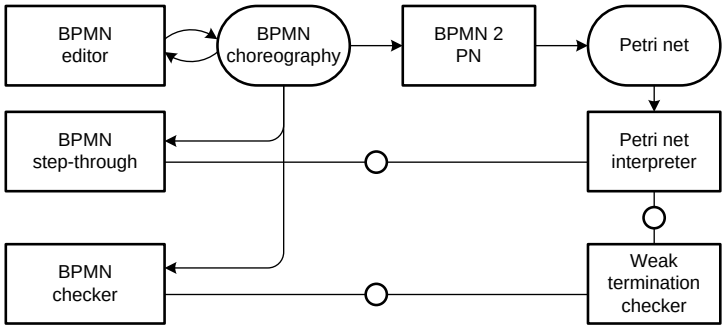


Figure 6.2: Tool chain for BPMN

The BPMN to Petri net converter produces Petri nets following the algorithm from [96]. This algorithm does not support the mapping of all BPMN constructs, e.g. complex gateways, OR gateways and multiple instances activities are ignored.

The resulting Petri net is used as input for a Petri net interpreter that identifies the enabled transitions in a given marking and fire transitions. This component is used by a BPMN step-through component that allows the modeler to simulate a BPMN diagram and therefore better

understand its control flow semantics. The enabled transitions in a given marking are projected back to the activities, gateways and events in the original BPMN diagram. Upon selection of the next activity or event to execute, the corresponding transition fires in the Petri net.

The BPMN checker is based on a weak termination checker and ensures the absence of deadlocks in BPMN diagrams. Weak termination was mentioned in the context of compatibility and desynchronizability checking (Section 2.5.1) and ensures that a valid final marking is reachable from every reachable marking. As valid final states are not defined in a BPMN diagram, the following workaround was chosen. Every BPMN end event is mapped to a place without outgoing edges. For each BPMN process it is assumed that it is desirable that at least one place will eventually contain one token and all places not originating from end events are empty. As a BPMN process might have multiple end events, the corresponding combinations of places imply the set of valid final markings.

In case the Petri net does not weakly terminate, this might be due to one of two reasons: (a) There is a deadlock. We can locate those transitions where at least one input place contains a token and highlight the corresponding elements in the BPMN diagram (typically AND gateways or intermediate message events). (b) There is lack of synchronization, i.e. the net is not 1-safe. Here, we can identify those transitions firing of which leads to unsafeness of the net. Again, we can highlight the corresponding BPMN element (typically XOR gateways or tasks with two incoming sequence flows). The software components and their connection are illustrated using the FMC block diagram notation [141] in Figure 6.2.

BPEL4Chor Implementation

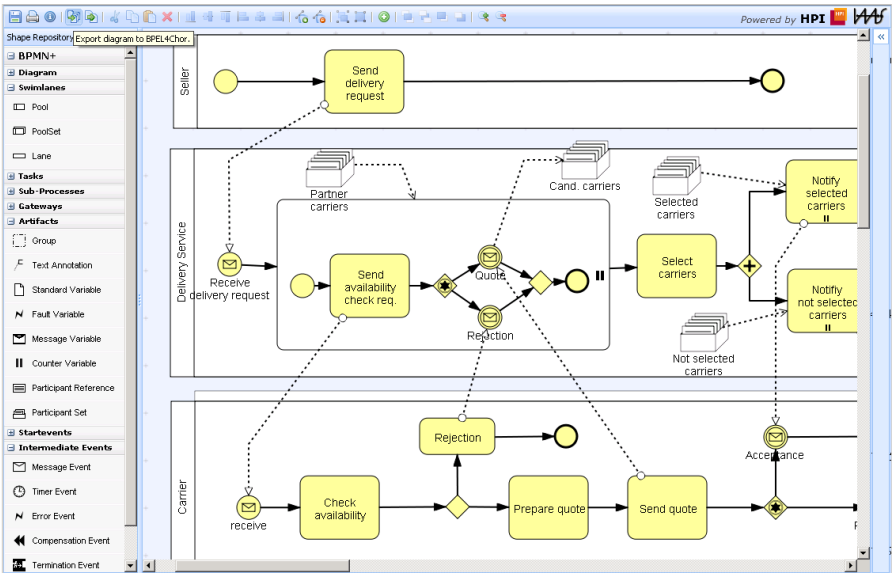


Figure 6.3: Extended BPMN that can be translated to BPEL4Chor

A Diploma thesis jointly supervised by the University of Stuttgart and the Hasso-Plattner-

Institute centered around the generation of BPEL4Chor out of BPMN choreographies. The main ideas behind the transformation were already explained in Section 4.3.1. The resulting prototype includes a stencil set for extended BPMN and the implementation of the transformation.

Figure 6.3 shows a screenshot of the prototype. The BPMN extensions in this implementation are based on BPMN 1.0 [6]. The transformation component returns a zip-file containing the topology-file and a set of BPEL-files for the participant behavior descriptions.

Instance Isolation Implementation

A stencil set for ν^*-nets and export functionality producing an extended version of PNML [41] were implemented. The PNML code can be read by a console tool providing a textual interface for stepping through ν^*-nets. Instance isolation checking was implemented as well.

```
<pnml><net>
 <place id="P1">
  <initialMarking>
   <token><name>x</name></token> <token><name>y</name></token>
  </initialMarking>
 </place>
 <transition id="T1" />
 <arc id="P1 to T1" source="P1" target="T1">
  <inscription><expression><var>a</var></expression></inscription>
 </arc>
 <arc id="T1 to P1" source="T1" target="P1">
  <inscription><expression><new/></expression></inscription>
 </arc>
</net></pnml>
```

Listing 14: PNML code for nu^*-nets

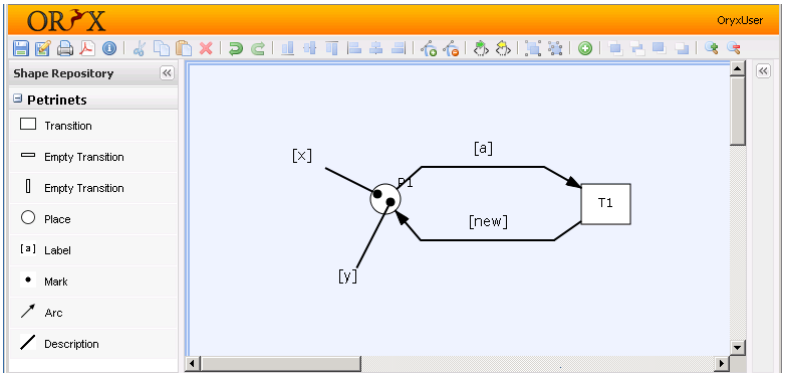


Figure 6.4: Screenshot of the ν^*-editor

Listing 14 shows the XML serialization for the ν^*-net from Figure 6.4. The net contains a place $P1$ and a transition $T1$. $P1$ contains two tokens carrying different names. The arcs representing the flow relationships have inscriptions attached with the corresponding variables.

Let's Dance Implementation

As Let's Dance was jointly developed with researchers of the Queensland University of Technology and SAP Research, the first tool implementation for Let's Dance was done at SAP in the form of Maestro for Let's Dance [72], an extension to the Maestro toolset of SAP Research.

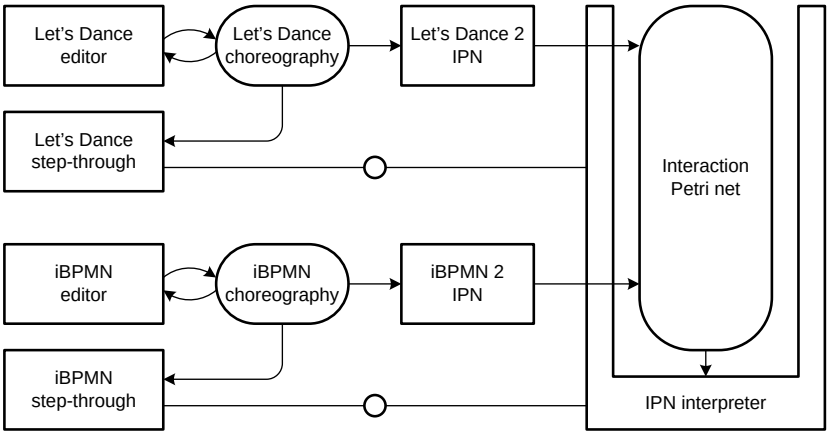


Figure 6.5: Tool chain for Let's Dance and iBPMN

A more recent prototype was implemented on top of Oryx, providing a graphical editor, a Let's Dance to interaction Petri nets transformator and step-through simulation functionality. Figure 6.5 shows the corresponding components using the FMC block diagram notation [141].

iBPMN Implementation

Similar functionality as it was already mentioned for Let's Dance is also available for iBPMN. Figure 6.5 enumerates the implemented software components. The mapping of iBPMN to interaction Petri nets was shown in Section 5.3.2. Figure 6.6 shows a screenshot of the iBPMN-editor. Including links to other diagrams or web resources is a central feature of Oryx. The collapsed complex interactions can have a diagram assigned that further refines it. Direct navigation is provided when clicking on the "+"-symbol. That way, hierarchical decomposition of choreographies is supported by the iBPMN editor. The shape repository on the left shows that pools, pool sets, the different kinds of interactions, message flow arcs, gateways, sequence flow arcs and events are the first-class citizens of iBPMN.

Realizability Implementation

Different flavors of realizability checking have been implemented. Full realizability checking, local enforceability checking and desynchronizability checking are directly accessible through the Oryx platform. The algorithms are based on the role projection algorithm as introduced in Section 5.1.3 and the composition mechanism introduced in the same chapter. Furthermore, full realizability and local enforceability use bi-simulation checking. Desynchronizability checking is based on weak termination checking that was already mentioned in Section 6. Figure 6.7

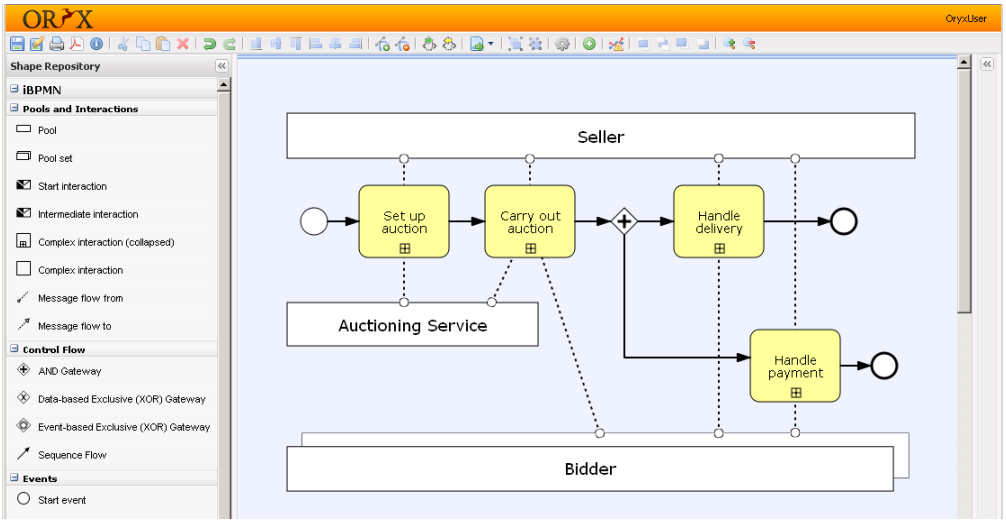


Figure 6.6: Screenshot of the iBPMN-editor

shows a screenshot of the interaction Petri net editor and the visualization of the desynchronizability checking functionality. The conflict transitions are highlighted.

Desynchronizability, full realizability and local enforceability checking are implemented as Java components that run as plugins in the Oryx backend. The serialization of the diagram under investigation is sent to the backend in RDF format. There, this format is parsed and translated into Java object structures. The algorithms then operate on these object structures and identify the conflict transitions in the case of desynchronizability checking. The overlay functionality of the Oryx Core is used to visualize the conflict transitions.

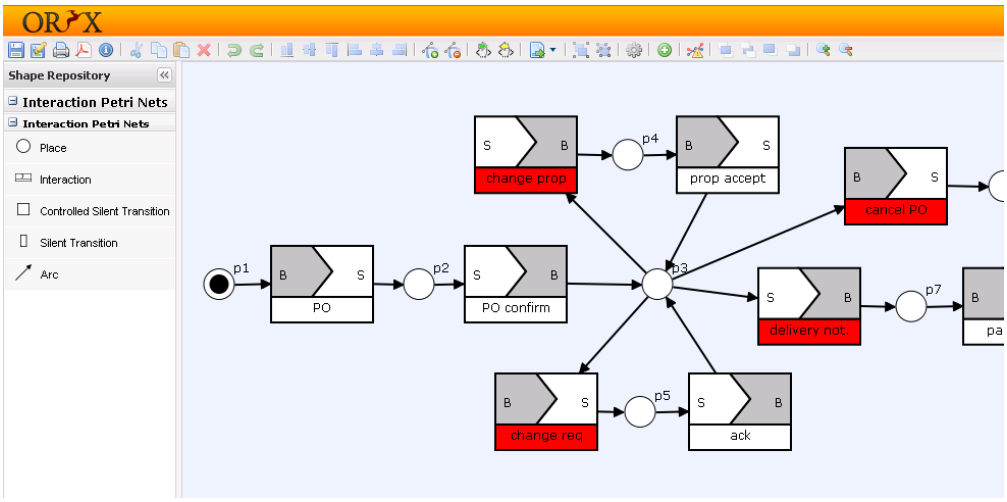


Figure 6.7: Screenshot of the interaction Petri net editor and desynchronizability checking

Chapter 7

Conclusion

This thesis has addressed two aspects of choreographies. On the one hand, it has centered around the suitability of choreography languages. On the other hand, it has investigated correctness criteria that are specific to choreographies.

Starting from the quest for suitable choreography languages, this thesis has surveyed prominent choreography languages that were put forward by industry initiatives. A set of requirements was identified that allows to assess choreography languages on the conceptual and on the technical level. These requirements were based on insights from existing choreography design approaches, from industry examples and from real-world system integration scenarios. They allow to compare the existing choreography languages and to identify white spots. The assessment revealed that BPMN and BPEL are good starting points for extensions. BPEL, being an orchestration language, needed to be shifted to a choreography language in the first place.

Starting from the assessment, two directions for language design were followed. (1) Minimal language extensions were applied to BPMN and BPEL, in order to overcome the limitations of these two languages. The resulting BPMN extensions and BPEL4Chor were validated using the requirements framework and an integration between these two languages was discussed. It was concluded that although mappings from one language to the other is possible in many cases, a complete round-tripping is unrealistic. (2) A more radical approach was followed by introducing the two novel choreography languages Let's Dance and iBPMN. The concepts behind Let's Dance and the notation used for this language are very different to the existing choreography languages. This resulted in low acceptance by modeling practitioners. iBPMN can be seen as compromise between radical change in semantics (shifting from an interconnection modeling style towards an interaction modeling style) and preserving the well-known notation of BPMN. The adoption of the notation increased iBPMN's acceptance.

Practical insights into the usage of interconnection modeling revealed that human modelers have difficulties with the interconnection modeling style and easily run into modeling errors. Interaction modeling largely overcomes this issue but introduces the new challenge of unrealizable choreographies. This underlines the fact that model quality plays a vital role also in the choreography space.

Therefore, the second area of investigation of this thesis is formal verification. Many use cases are already covered by existing verification techniques. Especially the notion of com-

patibility has been studied extensively in the literature. In contrast to this, the challenges of concurrent conversations and realizability were identified as focus areas for this thesis.

The notion of instance isolation was introduced as novel correctness criterion for choreographies, centering around proper correlation configurations. As part of that, ν^*-nets were introduced as formal model, inheriting strengths of open nets and π-calculus. Regarding interaction modeling, realizability of choreographies was a central theme. Based on interaction Petri nets, a novel formalism for interaction modeling including concurrency and decision ownership, realizability was studied in different flavors. These investigations finally led to full realizability, local enforceability and desynchronizability as novel correctness criteria for choreographies. In this context, it turned out that resolution strategies can have a big impact on the business semantics of choreographies and therefore fully automatic repair approaches are not suited.

Finally, prototypical implementations for the new choreography languages and verification techniques were presented. Especially the graphical editing capabilities of the tools enabled human modelers to experiment with the languages. This made it possible to more easily discuss the results with researchers all over the world and get students involved.

This thesis has introduced the explicit distinction between interconnection models and interaction models as a central theme. This distinction has spurred discussions in the academic community [142, 46]. Several other research groups have picked up the problem of realizability of interaction models and are currently investigating it further. Realizability is particularly interesting because it operates on a potentially infinite search space. While deadlock analysis, e.g. in the context of compatibility checking, is usually performed in finite state spaces, the number of potential participants that can realize a choreography is infinite.

This thesis has also influenced industrial initiatives. In particular, practitioners are adopting the interaction modeling style. A similar language like iBPMN will be part of the upcoming BPMN 2.0 standard in addition to the already existing interconnection modeling capabilities from BPMN 1.2. A dedicated choreography profile makes choreographies an essential part of BPMN 2.0. Also the role-based models of Let's Dance will be part of BPMN 2.0 under the name of "conversation models". This can be seen as further validation for the proposals made in this thesis. The advantages of interaction modeling have obviously made their way into industrial initiatives. The coming years will show how the co-existence of the two modeling styles in one language will be used in practice.

With the growing demand in cross-organizational integration of business processes and information systems, choreography modeling will likely increase in importance over the next years. The requirements framework, the languages and the correctness criteria presented in this thesis can serve as basis for advancing this field. However, a number of open issues remain and must be addressed in future work or by other researchers.

Open issues arise in the context of the four choreography language proposals and in the context of the formal verification techniques. The following sections will briefly discuss some of the issues.

Open Issues Regarding Choreography Languages

The BPMN extensions and BPEL4Chor cover large parts of the requirements framework. In contrast to this, Let's Dance and iBPMN still leave a number of white spots. Therefore, further extensions of these languages appear as obvious next step, a step that was not taken yet due to a number of new challenges that would arise immediately. The following explanations detail this.

As this thesis has rather focused on the language design along functional requirements, practical insights into the use of these languages also requires an in-depth empirical grounding, going beyond the results of this thesis.

Cancellation and Exception Handling in Interaction Models

Although cancellation was presented as an important requirement in Chapter 3, iBPMN only partially supports it. From a notational perspective this could easily be resolved by introducing attached intermediate events for sub-choreographies. Formal semantics could be defined similar to those presented in [96].

The main reason behind not integrating such cancellation features into iBPMN was that it would have caused serious issues regarding realizability. An interaction, a timeout or an exception that cancels a whole sub-choreography is typically only observable by one participant or by a subset of participants. That way, cancellation must ensure that interactions that were previously enabled are not enabled any longer. While in a synchronous and bi-lateral case this might be fully realizable, most asynchronous or multi-lateral scenarios will be unrealizable, or even if realizable desynchronizability will likely not be given.

It can be expected that the realizability notions as presented in Section 5.4 turn out to be too strict for the case of cancellation. It might be acceptable that certain interactions can still be carried out and their effect is ignored beyond a certain state of a conversation. This strategy is followed for orchestrations e.g. in the BPEL formalization by Lohmann [148]. However, due to the decentralized nature of choreographies such strategies are counterintuitive and must likely be disallowed. It might as well turn out that the autonomicity assumption behind the realizability notions is simply too demanding for the case of cancellation and that centralized coordination mechanisms for handling cancellation are needed in particular cases.

Other research groups, e.g. at the University of Stuttgart, are already investigating this issue, following an infrastructure-centric approach where an enterprise service bus (ESB) is extended into the direction of centralized exception handling in choreography settings [143].

Data Flow in Interaction Models

Interaction models as discussed in Chapter 5 mostly concentrate on behavioral dependencies, i.e. control flow. Extending interaction models with data flow capabilities is the consequent next step. This would also allow to introduce the notion of reference passing into interaction models. As it is the case with control flow, novel anomalies regarding data flow might arise. This is caused by the fact that the participants engaging in a conversation might have different assumptions about the current contents of data objects passed in interactions. Much like it is

possible to model unrealizable behavioral dependencies, it can likely turn out that it is possible to model unrealizable data flow dependencies.

Future work must investigate whether and how data flow modeling should be integrated with choreography modeling, which new anomalies can arise and how they can be detected and resolved.

Suitability of Languages and Empirical Studies

The requirements framework for choreography language focuses on functional requirements. However, this is only one ingredient for a "good" modeling language. As the practical experience with the choreography language Let's Dance shows, the acceptance of a modeling language cannot necessarily be traced back to the support of functional requirements.

BPMN, in contrast to Let's Dance, enjoys a major uptake in practice. This might be due to the fact that many big vendors are supporting and pushing the standard. Another main reason might be the wide acceptance by process modeling practitioners. This acceptance was not given in the case of Let's Dance. Especially the novel modeling constructs such as the Inhibits relationship were disliked by modelers. Also the fact that alternative branches could only be modeled implicitly, namely by different guarded interactions with opposing guard conditions, was disliked.

Measuring acceptance, which in turn might lead to the ultimate success of a language, can only be done through empirical studies. Also the questions of how well human modelers can use a modeling language to fulfill given tasks can be answered through empirical investigations. For instance, Jan Recker has empirically studied BPMN and other modeling languages in his PhD thesis [184]. In this context, experiments including read and write tests can deliver an understanding of how well human modelers perform in using one language in comparison to another [115]. Read tests concentrate on how well humans can answer questions based on a given model. Write tests focus on how well human modelers can create models using a certain modeling language.

Such empirical investigations were beyond the scope of this thesis. Future work must reveal empirical insights into the BPMN extensions, BPEL4Chor, Let's Dance and iBPMN. This is especially important for BPEL4Chor and iBPMN, as BPEL4Chor is a candidate for inclusion in standardization initiatives and parts of iBPMN have already been integrated into the BPMN 2.0 proposal [164] to a large extent.

Practical Insights into Incompatibility vs. Unrealizability

One of the motivations behind favoring interaction modeling was to avoid modeling anti-patterns. While this can actually be achieved through interaction modeling to a large extent, the new issue of unrealizability arises. It can be argued that being able to create unrealizable choreographies is not any better than being able to create incompatible choreographies.

By observing students that use both modeling styles it can be concluded that the probability of running into incompatibility is much higher than running into unrealizability. This is mostly due to the fact that simple structural rules of thumb such as "the sender of an interaction must

be the sender or receiver of all directly preceding interactions" can be applied locally and avoid many of the unrealizability problems.

Empirical evidence for this fact is missing and would need to be found through corresponding investigations. A first argument for the usefulness of the interaction modeling style can be derived from the fact that interaction modeling was introduced into the BPMN 2.0 proposal.

Open Issues Regarding Formal Verification

There do not only remain open issues regarding the choreography languages. Many further investigations in the direction of formal verification of choreographies are needed as well. This thesis is just a starting point for many interesting fields of research.

Investigations on ν^*-nets

Instance isolation checking was introduced in Section 4.4 on the basis of ν^*-nets. This formal model is very interesting in itself. The reachability and boundedness implications regarding simple Petri nets is one example. If a corresponding marking (or a transition) in a Petri net is not reachable then it can be implied that the marking (or transition) in the ν^*-nets is not reachable, either. If the corresponding Petri net is bounded, we can imply that the ν^*-net will also be bounded. This is due to the fact that a ν^*-net can be simulated by its corresponding Petri net.

Furthermore, ν^*-nets are likely to have the same expressiveness as π-calculus has. This could be shown by mapping π-calculus processes to ν^*-nets. Provided that this is possible, ν^*-nets would be Turing-complete. Therefore, ν^*-nets as simple and yet expressive formal model are an interesting object to study further.

A Unified Theory for Realizability

The role projection algorithm presented in Section 5.1 is related to the operating guidelines approach from [152]. This leads to the assumption that the issue of realizability is quite similar to the issue of controllability. Realizability is a criterion that is specific for interaction models. Controllability, on the contrary, is a criterion for individual open nets. Therefore, it is not obvious that these two fundamental criteria are related.

Different notions of controllability are available. Centralized controllability answers the question of whether one partner exists that can control the given open net. When partitioning the communication places into different ports, autonomous controllability and decentralized controllability can be distinguished: Autonomous controllability ensures that there exists a set of partners (one partner per port) that control the open net while not communicating among each other and not knowing the communication behavior of each other. Decentralized controllability ensures that there exists a set of partners that control the open net, do not communicate among each other but agree on the individual communication behavior.

In addition to the asynchronous case of open nets, controllability is also available for synchronous communication. This is important when applying controllability to the realizability

problem. Consider an interaction model as behavior model of a centralized observer of the interactions happening between the different participants. Each elementary interaction can now be considered to be an interaction between the sender, the observer and the receiver. Different ports can be identified based on the labeling of the interactions. Now, we check whether there exists a set of partners that control the behavior model of the observer. If this is the case, we know that at least a subset of the choreography can be realized by interacting partners.

Reusing controllability checking techniques for the realizability problem would allow to reuse a range of proofs and software tools that efficiently synthesize partners.

Future work must investigate the relationship between realizability and controllability in detail. The different notions available for controllability must be carefully matched with the different dimensions of realizability. This research work is currently carried out by the research group at the University of Rostock. First steps in this direction are already reported in [149].

Bibliography

[1] Unified Modeling Language Specification Version 1.4. Technical report, Object Management Group (OMG), 2001. http://www.omg.org/spec/UML/1.4/PDF.

[2] W3C Glossary and Dictionary. Technical report, W3C, 2003. http://www.w3.org/2003/glossary/.

[3] Open-EDI Reference Model, ISO/IEC JTC 1/SC30 ISO Standard 14662, Second Edition. Technical report, ISO, 2004.

[4] WS-Security Specification. URL: www.oasis-open.org/specs/index.php#wssv1.0, March 2004.

[5] UML 2.0 Superstructure Specification. Technical report, Object Management Group (OMG), August 2005.

[6] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. http://www.bpmn.org/.

[7] UMM Foundation Module V1.0. Technical report, United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), 2006.

[8] Directories for electronic data interchange for administration, commerce and transport. Technical report, United Nations Economic Commission for Europe (UNECE), 2008. http://www.unece.org/trade/untdid/welcome.htm.

[9] Business Process Modeling Notation (BPMN), Version 1.2. Technical report, Object Management Group (OMG), Feb 2009. http://www.omg.org/spec/BPMN/1.2/.

[10] W. M. P. v. d. Aalst. Don't go with the flow: web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.

[11] W. M. P. v. d. Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203.

[12] W. M. P. v. d. Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views – correctness-by-design for services. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 139–153, Brisbane, Australia, September 2008. Springer Verlag.

[13] W. M. P. v. d. Aalst, A. J. Mooij, C. Stahl, and K. Wolf. Service interaction: Patterns, formalization, and analysis. In *Formal Methods for Web Services (SFM)*, volume 5569, pages 42–88. Springer Verlag, April 2009.

[14] W. M. P. v. d. Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In *Proceedings 3rd International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4184 of *LNCS*, pages 1–23, Vienna, Austria, September 2006. Springer Verlag.

[15] W. M. P. v. d. Aalst, A. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *International Conference on Business Process Management (BPM)*, volume 2678 of *LNCS*, pages 1–12, Eindhoven, The Netherlands, June 2003. Springer Verlag.

[16] W. M. P. v. d. Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.

[17] W. M. P. v. d. Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[18] W. M. P. v. d. Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, January 2002.

[19] W. M. P. v. d. Aalst and A. Weijters. *Process-Aware Information Systems: Bridging People and Software through Process Technology*, chapter Process Mining, pages 235–255. Wiley & Sons, 2005.

[20] W. M. P. v. d. Aalst and M. Weske. The P2P Approach to Interorganizational Workflows. In *Proceedings of the 13th Conference on Advanced Information Systems Engineering (CAiSE)*, volume 2068 of *LNCS*, pages 140–156, Interlaken, Switzerland, June 2001. Springer Verlag.

[21] L. Aldred, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. On the notion of coupling in communication middleware. In *On the Move to Meaningful Internet Systems (OTM Conferences), Part II*, volume 3761 of *LNCS*, pages 1015–1033, Agia Napa, Cyprus, 2005. Springer Verlag.

[22] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proceedings of the 22nd international conference on Software engineering (ICSE)*, pages 304–313, New York, NY, USA, 2000. ACM.

[23] R. Alur, G. J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17:70–77, 1996.

[24] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *LNCS*, pages 114–129, London, UK, 1999. Springer Verlag.

[25] 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. Technical report, OASIS, May 2003. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`.

[26] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, June 2004.

[27] A. Arkin. Business Process Modeling Language (BPML). Technical report, Business Process Management Initiative (BPMI), 2002.

[28] A. Arkin et al. Web Service Choreography Interface (WSCI) 1.0. Technical report, Aug 2002. `http://www.w3.org/TR/wsci/`.

[29] J. Austin. *How to do Things with Words*. Harvard University Press, Cambridge, USA, 1955.

[30] A. Awad, G. Decker, and M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *Proceedings of the 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 326–341, Milan, Italy, September 2008. Springer Verlag.

[31] J. C. M. Baeten, editor. *Applications of process algebra*. Cambridge University Press, New York, NY, USA, 1990.

[32] C. Baier and J. P. Katoen. *Principles of Model Checking - The MIT Press*. 2008.

[33] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. *Web Services Conversation Language (WSCL) 1.0, W3C Note*, March 2002. `http://www.w3.org/TR/wscl10/`.

[34] A. Barros, G. Decker, and M. Dumas. Multi-staged and Multi-viewpoint Service Choreography Modelling. In *Proceedings Workshop on Software Engineering Methods for Service Oriented Architecture (SEMSOA)*, number 244 in CEUR Workshop Proceedings, Hannover, Germany, May 2007.

[35] A. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 4422 of *LNCS*, pages 245–259, Braga, Portugal, March 2007. Springer Verlag.

[36] A. Barros, M. Dumas, and A. ter Hofstede. Service Interaction Patterns. In *Proceedings 3rd International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 302–318, Nancy, France, 2005. Springer Verlag.

[37] A. P. Barros and G. Decker. Dynamic routing as paradigm for decentralized flexible process management. In *Proceedings of the Tenth IEEE International Enterprise Distributed Object Computing (EDOC) Workshops*, page 27, Hong Kong, China, October 2006. IEEE Computer Society.

[38] T. Basten and W. M. P. van der Aalst. Inheritance of behavior. *JLAP*, 47(2):47–145, 2001.

[39] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proceedings of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1217 of *LNCS*, pages 259–274, London, UK, 1997. Springer Verlag.

[40] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st international conference on Very large data bases (VLDB)*, pages 613–624. VLDB Endowment, 2005.

[41] J. Billington, S. Christensen, K. M. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *24th International Conference on the Applications and Theory of Petri Nets (ICATPN)*, volume 2679 of *LNCS*, pages 483–505, Eindhoven, The Netherlands, June 2003. Springer Verlag.

[42] L. Bocchi, Y. Hong, A. Lopes, and J. L. Fiadeiro. From BPEL to SRML: A Formal Transformational Approach. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 92–107, Brisbane, Australia, September 2007. Springer Verlag.

[43] E. Börger and B. Thalheim. *A Method for Verifiable and Validatable Business Process Modeling*, volume 5316 of *LNCS*, pages 59–115. Springer Verlag, 2008.

[44] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. Technical report, W3C, May 2000. `http://www.w3.org/TR/SOAP`.

[45] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[46] M. Bravetti and G. Zavattaro. Contract Compliance and Choreography Conformance in the Presence of Message Queues. In *Proceedings 5th Workshop on Web Services and Formal Methods (WS-FM)*, LNCS, Milan, Italy, 2008. Springer Verlag.

[47] S. Briais. ABC Bisimulation Checker, 2003. `http://lamp.epfl.ch/~sbriais/abc/abc.html`.

[48] T. Bultan and X. Fu. Choreography modeling and analysis with collaboration diagrams. *IEEE Data Eng. Bull.*, 31(3):27–30, 2008.

[49] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, April 2008.

[50] S. Burbeck. The tao of e-business services: The evolution of web applications into service-oriented components with web services, October 2000. `http://www.ibm.com/developerworks/webservices/library/ws-tao/`.

[51] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and Orchestration: A Synergic Approach for System Design. In *Proceedings 3rd International Conference on Service Oriented Computing (ICSOC)*, volume 3826 of *LNCS*, pages 228–240, Amsterdam, The Netherlands, Dec 2005. Springer Verlag.

[52] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and Orchestration Conformance for System Design. In *Proceedings 8th International Conference on Coordination Models and Languages (COORDINATION)*, volume 4038 of *LNCS*, pages 63–81, Bologna, Italy, June 2006. Springer Verlag.

[53] C. Canal, E. Pimentel, and J. M. Troya. Compatibility and inheritance in software architectures. *Sci. Comput. Program.*, 41(2):105–138, 2001.

[54] M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *Proceedings 16th European Symposium on Programming (ESOP) as part of the European Joint Conferences on Software Theory and Practice (ETAPS)*, volume 4421 of *LNCS*, pages 2–17, Braga, Portugal, March 2007. Springer Verlag.

[55] D. Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[56] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, Mar 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[57] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer. ebXML Business Process Specification Schema Version 1.01. Technical report, UN/CEFACT and OASIS, May 2001. `http://www.ebxml.org/specs/ebBPSS.pdf`.

[58] E. Colombo, J. Mylopoulos, and P. Spoletini. Modeling and analyzing context-aware composition of services. In *Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC)*, volume 3826 of *LNCS*, pages 198–213, Amsterdam, The Netherlands, December 2005. Springer Verlag.

[59] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, July 2001.

[60] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, and N. R. England. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, 80:315–325, 1997.

[61] M. Crawford et al. ebXML Core Components Technical Specification 2.01. Technical report, UN/CEFACT, November 2003.

[62] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, 2005.

[63] T. H. Davenport. *Process Innovation – Reengineering Work through Information Technology*. Havard Business School Press, 1992.

[64] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 109–120, New York, NY, USA, 2001. ACM Press.

[65] G. Decker. Choreographiemodellierung - Eine Übersicht. *Informatik Spektrum*, 31(2):161–166, Feb 2008.

[66] G. Decker. Realizability of Interaction Models. In *Proceedings Central European Workshop on Services and Their Composition (ZEUS)*, volume 438, Stuttgart, Germany, 2009. CEUR-WS.org.

[67] G. Decker. Tool erlaubt Gestaltung im Web. *Computer Zeitung*, 15:18, 2009.

[68] G. Decker and A. Barros. Interaction Modeling using BPMN. In *Proceedings 1st International Workshop on Collaborative Business Processes (CBP)*, number 4928 in LNCS, pages 206–217, Brisbane, Australia, September 2007. Springer Verlag.

[69] G. Decker, A. Barros, F. M. Kraft, and N. Lohmann. Non-desynchronizable service choreographies. In *Proceedings 6th International Conference on Service Oriented Computing (ICSOC)*, volume 5364 of *LNCS*, pages 331–346, Sydney, Australia, Dec 2008. Springer Verlag.

[70] G. Decker, R. Dijkman, M. Dumas, and L. Garcia-Banuelos. A Tool for Transforming BPMN to YAWL. In *Proceedings 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 386–389, Milan, Italy, Sep 2008. Springer Verlag.

[71] G. Decker, L. Gericke, S. Krumnow, and M. Weske. Prozessmodellierung und -ausführung im Web. In *Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA)*, Sankt Augustin, Germany, September 2008.

[72] G. Decker, M. Kirov, J. M. Zaha, and M. Dumas. Maestro for Let's Dance: An Environment for Modeling Service Interactions. In *Proceedings Demo Session 4th International Conference on Business Process Management (BPM)*, Vienna, Austria, 2006. CEUR-Workshop proceedings.

[73] G. Decker, O. Kopp, and A. Barros. An Introduction to Service Choreographies. *it - Information Technology*, 50:122–127, 2008.

[74] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske. Modeling Service Choreographies using BPMN and BPEL4Chor. In *Proceedings 20th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 5074 of *LNCS*, pages 79–93, Montpellier, France, June 2008. Springer Verlag.

[75] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *Proceedings IEEE 2007 International Conference on Web Services (ICWS)*, pages 296–303, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[76] G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting Services: From Specification to Execution. *Data & Knowledge Engineering*, 2009 (to appear).

[77] G. Decker, O. Kopp, and F. Puhlmann. Service Referrals in BPEL-based Choreographies. In *Proceedings of the 2nd European Young Researchers Workshop on Service Oriented Computing (YR-SOC)*, pages 25–30, Leicester, UK, June 2007. University of Leicester.

[78] G. Decker, A. Lueders, H. Overdick, K. Schlichting, and M. Weske. RESTful Petri Net Execution. In *Proceedings of the 5th Workshop on Web Services and Formal Methods (WS-FM)*, LNCS, Milan, Italy, Sep 2008. Springer Verlag.

[79] G. Decker and J. Mendling. Instantiation semantics for process models. In *Proceedings of the 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 164–179, Milan, Italy, Sep 2008. Springer Verlag.

[80] G. Decker and J. Mendling. Process instantiation. *Data & Knowledge Engineering*, 2009 (to appear).

[81] G. Decker, H. Overdick, and M. Weske. Oryx - An Open Modeling Platform for the BPM Community. In *Proceedings 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 382–385, Milan, Italy, Sept 2008. Springer Verlag.

[82] G. Decker, H. Overdick, and M. Weske. Oryx - sharing conceptual models on the web. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER)*, volume 5231 of *LNCS*, pages 536–537, Barcelona, Spain, October 2008. Springer Verlag.

[83] G. Decker, H. Overdick, and J. M. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA)*, volume P-95 of *LNI*, pages 21–34, Hamburg, Germany, October 2006. Gesellschaft für Informatik.

[84] G. Decker and F. Puhlmann. Extending BPMN for Modeling Complex Choreographies. In *Proceedings 15th International Conference on Cooperative Information Systems (CoopIS)*, volume 4803 of *LNCS*, pages 24–40, Vilamoura, Portugal, November 2007. Springer Verlag.

[85] G. Decker, F. Puhlmann, and M. Weske. Formalizing Service Interactions. In *Proceedings 4th International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 414–419, Vienna, Austria, Sept 2006. Springer Verlag.

[86] G. Decker and T. Schreiter. OMG releases BPMN 1.1 - What's changed? *EMISA Forum*, 28(2):12–20, 2008.

[87] G. Decker and M. von Riegen. Scenarios and techniques for choreography design. In *Proceedings of the 10th International Conference on Business Information Systems (BIS)*, volume 4439 of *LNCS*, pages 121–132, Poznan, Poland, April 2007. Springer Verlag.

[88] G. Decker and M. Weske. Behavioral Consistency for B2B Process Integration. In *Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 4495 of *LNCS*, pages 81–95, Trondheim, Norway, June 2007. Springer Verlag.

[89] G. Decker and M. Weske. Local Enforceability in Interaction Petri Nets. In *Proceedings 5th International Conference on Business Process Management (BPM)*, volume 4714 of *LNCS*, pages 305–319, Brisbane, Australia, September 2007. Springer Verlag.

[90] G. Decker and M. Weske. Instance Isolation Analysis for Service-Oriented Architectures. In *Proceedings of the IEEE 2008 International Conference on Services Computing (SCC)*, pages 249–256, Honolulu, USA, July 2008. IEEE Computer Society.

[91] G. Decker, J. M. Zaha, and M. Dumas. Execution Semantics for Service Choreographies. In *Proceedings 3rd International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4184 of *LNCS*, pages 163–177, Vienna, Austria, Sept 2006. Springer Verlag.

[92] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[93] J. L. Dietz. The deep structure of business processes. *Commun. ACM*, 49(5):58–64, 2006.

[94] J. L. G. Dietz. Understanding and modelling business processes with DEMO. In *Proceedings 18th International Conference on Conceptual Modeling (ER)*, volume 1728 of *LNCS*, pages 188–202, Paris, France, Nov 1999. Springer Verlag.

[95] R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.

[96] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology (IST)*, 2008.

[97] S. Dolev. *Self-stabilization*. MIT Press, 2000.

[98] J. Dorn, C. Grun, H. Werthner, and M. Zapletal. A Survey of B2B Methodologies and Technologies: From Business Models towards Deployment Artifacts. In *Proceedings 40th Hawaii International International Conference on Systems Science (HICSS)*, page 143, Waikoloa, Big Island, HI, USA, Jan 2007. IEEE Computer Society.

[99] J.-J. Dubray, S. S. Amand, and M. J. M. (Eds.). ebXML Business Process Specification Schema Technical Specification v2.0.4. Technical report, OASIS, December 2006.

[100] C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *Proceedings 25th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of *LNCS*, pages 103–115, Aalborg, Denmark, July 1998. Springer Verlag.

[101] J. Ebert and G. Engels. Specialization of Object Life Cycle Definitions. Fachbericht Informatik 19/95, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, 1995.

[102] G. Engels, A. Förster, R. Heckel, and S. Thöne. *Process-Aware Information Systems*, chapter Process Modeling using UML, pages 85–117. Wiley Publishing, New York, 2005.

[103] G. Engels and L. Groenewegen. Specifications of Coordinated Behaviour by SOCCA. In *Proceedings 3rd European Workshop on Software Process Technology (EWSPT)*, volume 772 of *LNCS*, pages 128–151, Villard de Lans, France, 1994. Springer Verlag.

[104] G. Engels, J. M. Küster, and L. Groenewegen. Consistent interaction of software components. *J. Integr. Des. Process Sci.*, 6(4):2–22, 2002.

[105] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. *SIGSOFT Softw. Eng. Notes*, 26(5):186–195, 2001.

[106] J. Esparza. Decidability and complexity of petri net problems - an introduction. In *In Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428, Dagstuhl, Germany, 1998. Springer Verlag.

[107] D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative Control Flow. In *Proceedings of the 12th International Workshop on Abstract State Machines (ASM)*, pages 131–151. Paris XII, Mar. 2005.

[108] D. C. Fallside and P. Walmsley. Web Services Business Process Execution Language Version 2.0. Technical report, Oct 2005. `http://www.oasis-open.org/apps/org/workgroup/wsbpel/`.

[109] D. F. Ferguson and M. Stockton. Enterprise Business Process Management – Architecture, Technology and Standards. In *Proceedings 4th International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 1–15, Vienna, Austria, 2006. Springer Verlag.

[110] O. K. Ferstl and E. J. Sinz. *Foundations of Information Systems (in German), 5. Edition*. Oldenbourg, 2006.

[111] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society, June 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[112] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[113] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and analysis of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, November 2004.

[114] Y. Gao. BPMN-BPEL Transformation and Round Trip Engineering. Technical report, eClarus Software, 2006.

[115] A. Gemino and Y. Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requirements Engineering*, 9:248–260, 2004.

[116] A. Grosskopf. xBPMN. formal control flow specification of a BPMN based process execution language. Master's thesis, Hasso-Plattner-Institute, Potsdam, Germany, 2007.

[117] A. Grosskopf, G. Decker, and M. Weske. *The Process - Process Modeling using BPMN*. Meghan-Kiffer Press, 2009.

[118] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - Core, W3C Recommendation. Technical report, May 2006. `http://www.w3.org/TR/ws-addr-core/`.

[119] S. Gupti and S. Mukhi. *XML Schema Definition (XSD)*. BPB Publications.

[120] Y. Gurevich. *Specification and validation methods*, chapter Evolving algebras 1993: Lipari guide, pages 9–36. Oxford University Press, Inc., New York, NY, USA, 1995.

[121] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness, April 1994.

[122] T. Hettel, C. Flender, and A. P. Barros. Scaling Choreography Modelling for B2B Value-Chain Analysis. In *Proceedings 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 294–309, Milan, Italy, Sept 2008. Springer Verlag.

[123] J. Hidders, M. Dumas, W. M. van der Aalst, A. H. ter Hofstede, and J. Verelst. When Are Two Workflows the Same? In *Proceedings 11th Australasian Theory Symposium (CATS)*, pages 3–11, Newcastle, Australia, 2005. ACM.

[124] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, New York, 1985.

[125] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal. Deriving executable BPEL from UMM Business Transactions. In *IEEE International Conference on Services Computing (SCC)*, pages 178–186, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[126] A. H. M. T. Hofstede and T. P. V. D. Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology (IST)*, 1992.

[127] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[128] D. Hollingsworth. The workflow reference model. Technical report, Workflow Management Coalition, January 1995. `http://www.wfmc.org/standards/docs/tc003v11.pdf`.

[129] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition, November 2000.

[130] M. Ibrahim and G. Long. Service-Oriented Architecture and Enterprise Architecture, Part 1. Technical report, IBM, April 2007. `http://www.ibm.com/developerworks/webservices/library/ws-soa-enterprise1/`.

[131] ITU-T. Message sequence chart. Recommendation Z.120, ITU-T, 2000.

[132] ITU-T/ISO. Open distributed processing reference model. Technical Report ITU-T X.901..904 – ISO/IEC 10746-1.4, ITU-T/ISO, 1994–1997.

[133] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1*. Springer Verlag, 1996.

[134] a. A. Jo L. Bocchi, J. L. Fiadeiro, and A. Lopes. Specifying and Composing Interaction Protocols for Service-Oriented System Modelling. In *Proceedings of the 27th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, volume 4574 of *LNCS*, pages 358–373, Berlin, Heidelberg, 2007. Springer Verlag.

[135] S. Jones. *Enterprise SOA Adoption Strategies*. InfoQ Enterprise Software Development Series, 2006. `http://www.infoq.com/minibooks/enterprise-soa`.

[136] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report, November 2005. `http://www.w3.org/TR/ws-cdl-10`.

[137] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany, 1992.

[138] R. Khalaf. From RosettaNet PIPs to BPEL processes: A three level approach for business protocols. *Data & Knowledge Engineering*, 61:23–38, Apr. 2006.

[139] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, 2002.

[140] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for Sub-processes – BPEL-SPE. Technical report, IBM and SAP, 2005.

[141] A. Knöpfel, B. Gröne, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, May 2006.

[142] O. Kopp and F. Leymann. Choreography Design Using WS-BPEL. *IEEE Data Eng. Bull.*, 31(3):31–34, 2008.

[143] O. Kopp, M. Wieland, and F. Leymann. Towards Choreography Transactions. In *Proceedings Central European Workshop on Services and Their Composition (ZEUS)*, volume 438, Stuttgart, Germany, 2009. CEUR-WS.org.

[144] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[145] S. Krumnow, G. Decker, and M. Weske. Modellierung von EPKs im Web mit Oryx. In *MobIS Workshops*, volume 420 of *CEUR Workshop Proceedings*, pages 5–17. CEUR-WS.org, 2008.

[146] H. Kuno, M. Lemon, A. Karp, and D. Beringer. Conversations + Interfaces = Business Logic. Technical Report HPL-2001-127/20010601, Software Technology Laboratory, HP Laboratories, Palo Alto, CA, 2001.

[147] F. Leymann. Web Services Flow Language (WSFL 1.0), May 2001.

[148] N. Lohmann. A Feature-complete Petri Net Semantics for WS-BPEL 2.0. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 77–91, Brisbane, Australia, September 2007. Springer Verlag.

[149] N. Lohmann. Realizability is Controllability. In *Proceedings Central European Workshop on Services and Their Composition (ZEUS)*, volume 438, Stuttgart, Germany, 2009. CEUR-WS.org.

[150] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig. Analyzing BPEL4Chor: Verification and Participant Synthesis. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 46–60, Brisbane, Australia, September 2007. Springer Verlag.

[151] N. Lohmann, P. Massuthe, and K. Wolf. Behavioral constraints for services. In *Proceedings of the 5th International Conference on Business Process Management (BPM)*, volume 4714 of *LNCS*, pages 271–287, Brisbane, Australia, Sept. 2007. Springer Verlag.

[152] N. Lohmann, P. Massuthe, and K. Wolf. Operating guidelines for finite-state services. In *Proceedings of the 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN)*, volume 4546 of *LNCS*, pages 321–341, Siedlce, Poland, June 2007. Springer Verlag.

[153] N. Lohmann, E. Verbeek, and R. Dijkman. Petri net transformations for business processes - a survey. *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, 2008.

[154] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151, New York, NY, USA, 1987. ACM.

[155] A. Martens. Analyzing Web Service based Business Processes. In *Proceedings International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 3442 of *LNCS*, pages 19–33, Edinburgh, Scotland, April 2005. Springer Verlag.

[156] A. Martens. Consistency between Executable and Abstract Processes. In *Proceedings IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE)*, pages 60–67, Hong Kong, China, March 2005. IEEE Computer Society.

[157] P. Massuthe and K. Schmidt. Operating guidelines - an automata-theoretic foundation for the service-oriented architecture. In *Proceedings Fifth International Conference on Quality Software (QSIC)*, pages 452–457, Washington, DC, USA, 2005. IEEE Computer Society.

[158] M. Mazzara and R. Lucchi. pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, 70:96–118, 2006.

[159] J. Mendling and M. Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *Proceedings of OTM 2005 Workshops*, volume 3762 of *LNCS*, pages 506–515, Agia Napa, Cyprus, October 2005. Springer Verlag.

[160] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, 100:1–40, 1992.

[161] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann. BPEL for Semantic Web Services (BPEL4SWS). In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, volume 4805 of *LNCS*, pages 179–188, Vilamoura, Portugal, 2007. Springer Verlag.

[162] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann. BPELlight. In *Proceedings 5th International Conference on Business Process Management (BPM)*, volume 4714 of *LNCS*, pages 214–229, Brisbane, Australia, Sept 2007. Springer Verlag.

[163] J. Nitzsche, T. van Lessen, and F. Leymann. WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities. In *Proceedings 3rd International Conference on Internet and Web Applications and Services (ICIW)*, pages 168–173, Athens, Greece, June 2008. IEEE Computer Society.

[164] OMG. BPMN 2.0 RFP revised submission. Technical report, Object Management Group (OMG), Sep 2008. `http://www.omg.org/docs/bmi/08-09-04.pdf`.

[165] Organization for the Advancement of Structured Information Standards (OASIS). *SCA Assembly Model Specification V1.00*, March 2007.

[166] Organization for the Advancement of Structured Information Standards (OASIS). *SCA Client and Implementation Model Specification for WS-BPEL*, March 2007.

[167] C. Ouyang, M. Dumas, Ter, and W. M. P. van der Aalst. From BPMN Process Models to BPEL Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 285–292, Chicago, Illinois, USA, 2006. IEEE Computer Society.

[168] C. Ouyang, M. Dumas, A. H. ter Hofstede, and W. M. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research (JWSR)*, 2007.

[169] H. Overdick, F. Puhlmann, and M. Weske. Towards a formal model for agile service discovery and integration. In *Proceedings of the International Workshop on Dynamic Web Processes (DWP)*, volume RC23822 of *Technical Report*. IBM, 2005.

[170] M. P. Papazoglou. *Web Services: Principles and Technology*. Pearson, 2007.

[171] W. D. Pauw, R. Hoch, and Y. Huang. Discovering conversations in web services using semantic correlation analysis. In *Proceedings IEEE International Conference on Web Services (ICWS)*, pages 639–646, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[172] W. Peng. Single-link and time communicating finite state machines. In *Proceedings of 1994 International Conference on Network Protocols (ICNP)*, pages 126–133, Boston, USA, October 1994.

[173] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proceedings of 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 287–300, Annapolis, Maryland, Oct 2007. IEEE Computer Society.

[174] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In *Proceedings 15th International Conference on Coopartive Information Systems (CoopIS)*, volume 4803 of *LNCS*, pages 77–94, Vilamoura, Portugal, 2007. Springer Verlag.

[175] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[176] K. Pfitzner, G. Decker, O. Kopp, and F. Leymann. Web Service Choreography Configurations for BPMN. In *Proceedings 3rd International Workshop on Engineering Service-oriented Applications: Analysis, Design and Composition (WESOA)*, volume 4907 of *LNCS*, pages 401–412, Vienna, Austria, September 2007. Springer Verlag.

[177] G. Plotkin and V. Pratt. Teams can see pomsets extended abstract, 1990.

[178] F. Puhlmann. *On the Application of a Theory for Mobile Systems to Business Process Management*. Doctoral thesis, University of Potsdam, Potsdam, Germany, July 2007.

[179] F. Puhlmann and M. Weske. Using the Pi-Calculus for Formalizing Workflow Patterns. In *Proceedings 3rd International Conference on Business Process Management (BPM)*, volume 3649 of *LNCS*, pages 153–168, Nancy, France, 2005. Springer Verlag.

[180] F. Puhlmann and M. Weske. Interaction Soundness for Service Orchestrations. In *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC)*, volume 4294 of *LNCS*, pages 302–313, Chicago, IL, USA, December 2006. Springer Verlag.

[181] F. Puhlmann and M. Weske. Investigations on soundness regarding lazy activities. In *Proceedings 4th International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 145–160, Vienna, Austria, September 2006. Springer Verlag.

[182] Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *Proceedings 16th international conference on World Wide Web (WWW)*, pages 973–982, New York, NY, USA, 2007. ACM.

[183] D. Quartel, R. Dijkman, and M. van Sinderen. Methodological support for service-oriented design with ISDL. In *Proceedings of the 2nd Intenational Conference on Service-Oriented Computing (ICSOC)*, pages 1–10, New York City, NY, USA, November 2004. ACM.

[184] J. Recker. *Understanding Process Modelling Grammar Continuance*. PhD thesis, School of Information Systems at the Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, 2008.

[185] J. Recker and J. Mendling. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In *Proceedings of the 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD)*, June 2006.

[186] M. Reichert, S. Rinderle, U. Kreher, H. Acker, M. Lauer, and P. Dadam. ADEPT Next Generation Process Management Technology. In *CAiSE Forum*, volume 231. CEUR-WS.org, 2006.

[187] P. Reimann, O. Kopp, G. Decker, and F. Leymann. Generating WS-BPEL 2.0 Processes from a Grounded BPEL4Chor Choreography. Technischer Bericht Informatik 2008/07, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, October 2008.

[188] W. Reisig. *Petri nets*. Springer Verlag, 1985.

[189] E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[190] D. Sangiorgi. A Theory of Bisimulation for the pi-Calculus. *Acta Informatica*, 16(33):69–97, 1996.

[191] A.-W. Scheer and K. Schneider. *Handbook on Architectures of Information Systems*, chapter ARIS – Architecture of Integrated Information Systems, pages 605–623. Springer Verlag, 2005.

[192] A.-W. Scheer, O. Thomas, and O. Adam. *Process-Aware Information Systems*, chapter Process Modeling Using Event-Driven Process Chains. Wiley Publishing, New York, 2005.

[193] K. Schmidt. LoLA: A Low Level Analyser. In *Proceedings 21st International Conference on the Application and Theory of Petri Nets (ICATPN)*, volume 1825 of *LNCS*, pages 465–474, Aarhus, Denmark, 2000. Springer Verlag.

[194] K. Schmidt. Controllability of Open Workflow Nets. In *Enterprise Modelling and Information Systems Architectures (EMISA)*, volume P-75 of *LNI*, pages 236–249, Klagenfurt, Austria, 2005. Köllen Druck+Verlag GmbH.

[195] T. Schreiter. Towards Executability of BPMN: Data Perspective and Process Instantiation. Master's thesis, Hasso-Plattner-Institute, Potsdam, Germany, 2008.

[196] J. Searle. *Speech Acts*. Cambridge University Press, 1969.

[197] C. Seel and D. Vanderhaeghen. Meta-Model based Extensions of the EPC for Inter-Organisational Process Modelling. In *Proceedings 4th Workshop on Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK)*, volume 167 of *CEUR*, pages 117–136, Hamburg, Germany, December 2005.

[198] B. Selic. Using UML for Modeling Complex Real-Time Systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, volume 1474 of *LNCS*, pages 250–260, London, UK, 1998. Springer Verlag.

[199] H. Smith and P. Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2003.

[200] J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 1–16, Brisbane, Australia, Sept 2007. Springer Verlag.

[201] P. Tarr, H. Ossher, W. Harrison, and J. Sutton, Stanley M. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering (ICSE)*, pages 107–119, New York, NY, USA, 1999. ACM.

[202] S. Tasharofi, M. Vakilian, R. Z. Moghaddam, and M. Sirjani. Modeling Web Service Interactions Using the Coordination Language Reo. In *Proceedings 4th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 4937 of *LNCS*, pages 108–123, Brisbane, Australia, September 2007. Springer Verlag.

[203] S. Thatte. XLANG Web Services for Business Process Design. Technical report, Microsoft Corporation, 2001.

[204] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Proceedings of International Conference on the Applications and Theory of Petri Nets (ICATPN)*, volume 3536 of *LNCS*, pages 444–454, Miami, USA, 2005. Springer Verlag.

[205] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[206] K. M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In *Proceedings 24th International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 2679 of *LNCS*, pages 337–356, Eindhoven, The Netherlands, 2003. Springer Verlag.

[207] K. M. van Hee, N. Sidorova, and M. Voorhoeve. Generalised soundness of workflow nets is decidable. In *Proceedings 25th International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 3099 of *LNCS*, pages 197–215, Bologna, Italy, 2004. Springer Verlag.

[208] F. R. Velardo and D. de Frutos-Escrig. Name creation vs. replication in petri net systems. In *Proceedings 28th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (Petri Nets)*, volume 4546 of *LNCS*, pages 402–422, Siedlce, Poland, June 2007. Springer Verlag.

[209] B. Victor, F. Moller, M. Dam, and L.-H. Eriksson. The Mobility Workbench, 2005. `http://www.it.uu.se/research/group/mobility/mwb`.

[210] W3C. Extensible markup language (xml). `http://www.w3.org/XML/`.

[211] A. E. Walsh, editor. *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference, 2002.

[212] M. Weidlich, G. Decker, A. Grosskopf, and M. Weske. BPEL to BPMN: The Myth of a Straight-Forward Mapping. In *Proceedings 16th International Conference on Cooperative Information Systems (CoopIS)*, volume 5331 of *LNCS*, pages 265–282, Monterrey, Mexico, Nov 2008. Springer Verlag.

[213] M. Weidlich, G. Decker, and M. Weske. Efficient analysis of bpel 2.0 processes using pi-calculus. In *Proceedings of the The 2nd IEEE Asia-Pacific Service Computing Conference (APSCC)*, pages 266–274, Washington, DC, USA, 2007. IEEE Computer Society.

[214] M. Weske. *Business Process Management: Concepts, Languages, Architectures* . Springer Verlag, 2007.

[215] S. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, 2005.

[216] P. Wohed, W. M. van der Aalst, M. Dumas, A. ter Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Proceedings 4th International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 161–176, Vienna, Austria, 2006. Springer Verlag.

[217] P. Wohed, W. M. P. van der Aalst, M. Dumas, and T. A. H. M. Hofstede. Analysis of web services composition languages: The case of BPEL4WS. In *Proceedings 22nd International Conference on Conceptual Modeling (ER)*, volume 2813 of *LNCS*, pages 200–215, Chicago, Illinois, USA, 2003. Springer Verlag.

[218] P. Y. Wong and J. Gibbons. A Process Semantics for BPMN. In *Proceedings of 10th International Conference on Formal Engineering Methods (ICFEM)*, volume 5256 of *LNCS*, pages 355–374, Kitakyushu-City, Japan, October 2008. Springer Verlag.

[219] D. Woods. *Enterprise Services Architecture*. O'Reilly, 2003.

[220] J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker. Service Interaction Modeling: Bridging Global and Local Views. In *Proceedings 10th IEEE International EDOC Conference (EDOC)*, pages 45–55, Hong Kong, Oct 2006. IEEE Computer Society.

[221] J. M. Zaha, M. Dumas, A. H. ter Hofstede, A. Barros, and G. Decker. Bridging Global and Local Models of Service-oriented Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(3):302–318, 2008.

[222] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson. Developing web services choreography standards: the case of REST vs. SOAP. *Decis. Support Syst.*, 40(1):9–29, 2005.

[223] M. zur Muehlen and J. Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *Proceedings 20th International Conference Advanced Information Systems Engineering (CAiSE)*, volume 5074 of *LNCS*, pages 465–479, Montpellier, France, June 2008. Springer Verlag.

All web links were last followed on 20 June 2009.