Standard for the Subject-oriented Specification of Systems

Albert Fleischmann *Editor*

Standard for the Subject-oriented Specification of Systems

Working Document

Egon Börger
xyz

Stefan Borgert
xyz

Matthes Elstermann
xyz

Albert Fleischmann
xyz

Reinhard Gniza
xyz

Herbert Kindermann
xyz

Florian Krenn
xyz

Werner Schmidt
xyz

Robert Singer
FH JOANNEUM–University of Applied Sciences

Christian Stary
xyz

Florian Strecker
xyz

André Wolski
xyz

Typeset by the authors

Production and publishing: XYZ

ISBN: 978-3-123-45678-9 (dummy)

Short contents

Preface

Contents

CHAPTER 1

Foundation

To facilitate the understanding of the following sections we will introduce the philosophy of subject-orienting modeling which is based on the Parallel Activity Specification Scheme (PASS). Additional, we will give a short introduction to ontologies—especially the Web Ontology Language (OWL)—, and to Abstract State Machines (ASM) as underlying concepts of this standard document.

1.1 Subject Orientation and PASS

In this section, we lay the ground for PASS as a language for describing processes in a subject-oriented way. This section is not a complete description of all PASS features, but it gives the first impression about subject-orientation and the specification language PASS. The detailed concepts are defined in the upcoming chapters.

The term subject has manifold meanings depending on the discipline. In philosophy, a subject is an observer and an object is a thing observed. In the grammar of many languages, the term subject has a slightly different meaning. "According to the traditional view, the subject is the doer of the action (actor) or the element that expresses what the sentence is about (topic)." [Kee76]. In PASS the term subject corresponds to the doer of an action whereas in ontology description languages, like RDF (see section 1.2), the term subject means the topic what the "sentence" is about.

1.1.1 Subject-driven Business Processes

Subjects represent the behavior of an active entity. A specification of a subject does not say anything about the technology used to execute the described behavior. This is different to other encapsulation approaches, such as multi-agent systems.

Subjects communicate with each other by exchanging messages. Messages have a name and a payload. The name should express the meaning of a message informally and the payloads are the data (business objects) transported. Internally, subjects execute local activities such as calculating a price, storing an address, etc.

A subject sends messages to other subjects, expects messages from other subjects, and executes internal actions. All these activities are done in sequences which are defined in a subject's behavior specification. Subject-oriented process specifications are always embedded in a context. A context is defined by the business organization and the technology by which a business process is executed.

Subject-oriented system development integrates established theories and concepts. It has been inspired by various process algebras (see e.g. [2], [3], [4]), by the basic structure of nearly all natural languages (Subject, Predicate, Object) and the systemic sociology developed by Niklas Luhmann (an introduction can be found in [5]). According to the organizational theory developed by Luhmann, the smallest organization consists of communication executed between at least two information processing entities [5]. The integrated concepts have been enhanced and adapted to organizational stakeholder requirements, such as providing a simple graphical notation, as detailed in the following sections.

1.1.2 Subject Interaction and Behavior

We introduce the basic concepts of process modeling in S-BPM using a simple order process. A customer sends an order to the order handling department of a supplier. He is going to receive an order confirmation and the ordered product by the shipment company. Figure 1.3 shows the communication structure of that process. The involved subjects and the messages they exchange can easily be grasped.

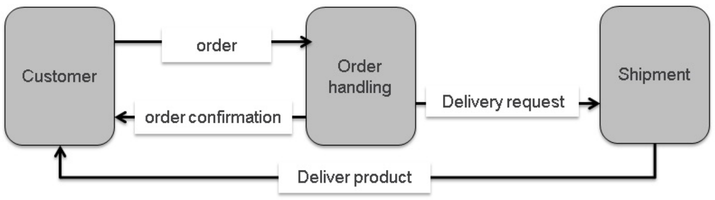

Figure 1.1: The Communication Structure in the Order Process

Each subject has a so-called input pool which is its mailbox for receiving messages. This input pool can be structured according to the business requirements at hand. The modeler can define how many messages of which type and/or from which sender can be deposited and what the reaction is if these restrictions are violated. This means the synchronization through message exchange can be specified for each subject individually.

Messages have an intuitive meaning expressed by their name. A formal semantics is given by their use and the data which are transported with a message. Figure 1.2 depicts the behavior of the subjects "customer" and "order handling".

In the first state of its behavior, the subject "customer" executes the internal function "Prepare order". When this function is finished the transition "order prepared" follows. In the succeeding state "send order" the message "order" is sent to the subject "order handling". After this message is sent (deposited in the input pool of subject "order handling"), the subject "Customer" goes into the state "wait for confirmation". If this message is not in the input pool the subject stops its execution until the corresponding message arrives in the input pool. On arrival, the subject removes the message from the input pool and follows the transition into state "Wait for product" and so on.

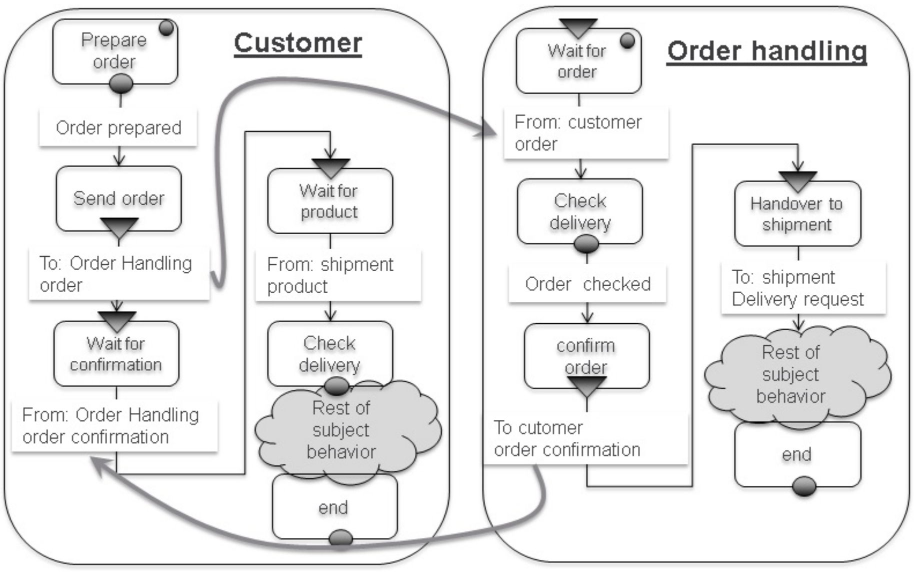

Figure 1.2: The Behavior of Subjects

The subject "Order Handling" waits for the message "order" from the subject "customer". If this message is in the input pool it is removed and the succeeding function "check order" is executed and so on.

The behavior of each subject describes in which order it sends messages, expects (receives) and performs internal functions. Messages transport data from the sending to the receiving subject and internal functions operate on internal data of a subject. These data aspects of a subject are described in section 1.1.3. In a dynamic and fast-changing world, processes need to be able to capture known but unpredictable events. In our example let us assume that a customer can change an order. This means the subject "customer" may send the message "Change order" at any time. Figure 1.3 shows the corresponding communication structure, which now contains the message "change order".

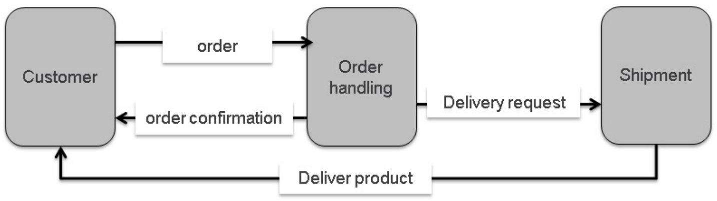

Figure 1.3: The Communication Structure with Change Message

Due to this unpredictable event, the behavior of the involved subjects needs also to be adapted. Figure 1.4 illustrates the respective behavior of the customer.

The subject "customer" may have the idea to change its order in the state "wait for confirmation" or in the state "wait for product". The flags in these states indicate that there is a so-called behavior extension described by a so-called non-deterministic event guard [12, 22]. The non-deterministic event created in the subject is the idea "change order". If this idea comes up, the current states, either "wait for confirmation" or "wait for product", are left, and the subject "customer" jumps into state "change order" in the guard behavior. In this state, the message

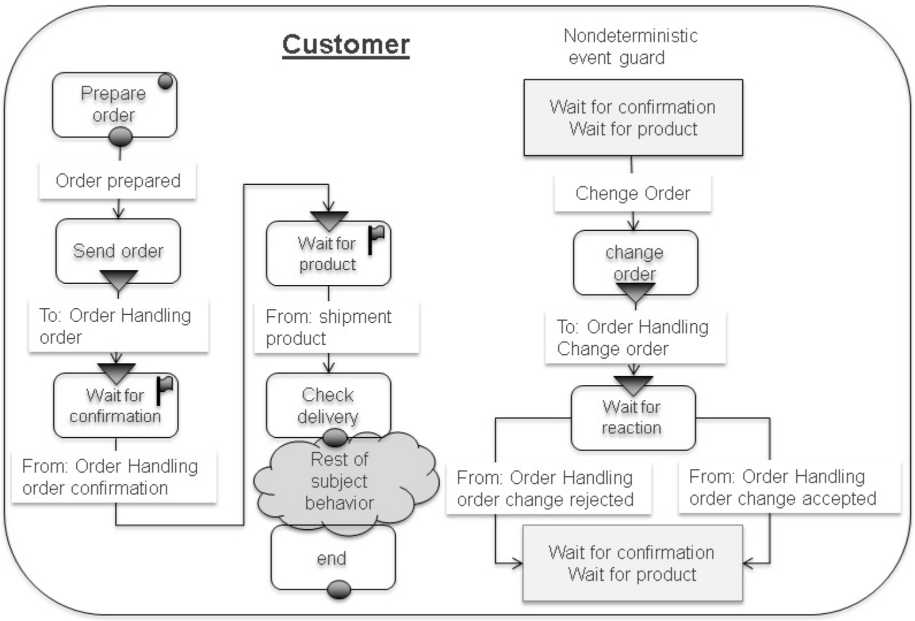
Customer
Nondeterministic event guard
Prepare order
Order prepared
Send order
To: Order Handling order
Wait for confirmation
From: Order Handling order confirmation
Wait for product
From: shipment product
Check delivery
Rest of subject behavior
end
Wait for confirmation Wait for product
Chenge Order
change order
To: Order Handling Change order
Wait for reaction
From: Order Handling order change rejected
From: Order Handling order change accepted
Wait for confirmation Wait for product

Figure 1.4: Customer is allowed to Change Orders

"change order" is sent and the subject waits in the state "wait for reaction". In this state, the answer can either be "order change accepted" or "order change rejected". Independently of the received message the subject "customer" moves to the state "wait for product". The message "order change accepted" is considered as confirmation, if a confirmation has not arrived yet (state "wait for confirmation"). If the change is rejected the customer has to wait for the product(s) he/she has ordered originally. Similar to the behavior of the subject "customer" the behavior of the subject "order handling" has to be adapted.

1.1.3 Subjects and Objects

Up to now, we did not mention data or the objects with their predicates, to get complete sentences comprising subject, predicate, and object. Figure 1.1.3 displays how subjects and objects are connected. The internal function "prepare order" uses internal data to prepare the data for the order message. This order data is sent as the payload of the message "order".

The internal functions in a subject can be realized as methods of an object or functions implemented in a service if a service-oriented architecture is available. These objects have an additional method for each message. If a message is sent, the method allows receiving data values sent with the message, and if a message is received the corresponding method is used to store the received data in the object [22]. This means either subject are the entities which use synchronous services as an implementation of functions or asynchronous services are implemented through subjects or even through complex processes consisting of several subjects. Consequently, the concept Service Oriented Architecture (SOA) is complementary to S-BPM: Subjects are the entities which use the services offered by SOAs (cf. [25]).

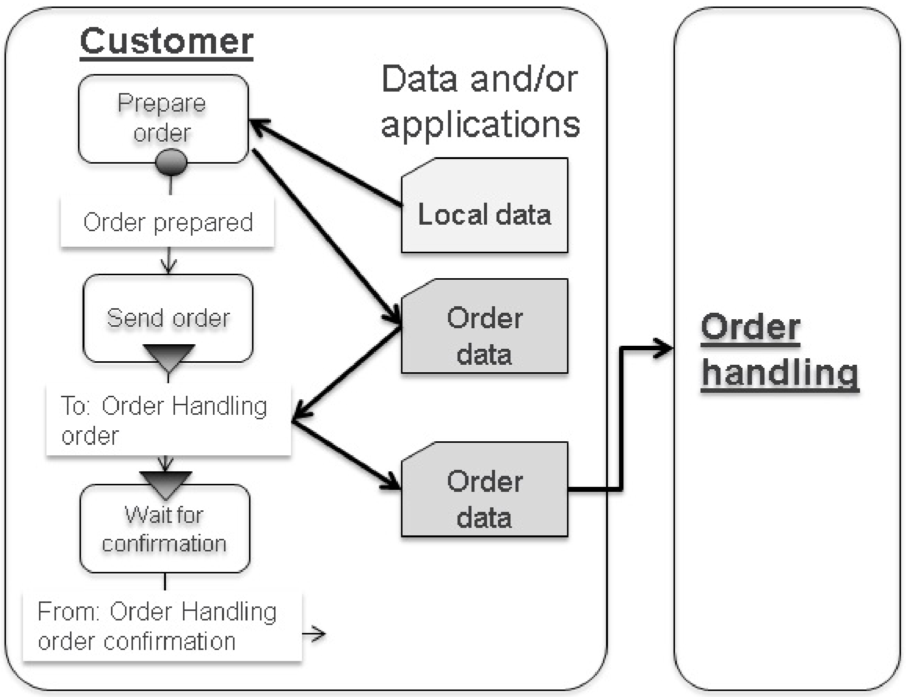

Figure 1.5: Subjects and Objects

1.2 Introduction to Ontologies and OWL

This short introduction to ontology, the Resource Description Framework and Web Ontology Language (OWL), should help to get an understanding of the PASS ontology outlined in section 2 and **??**.

Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects of classes.

In computer science and information science, an ontology encompasses a representation, formal naming, and definition of the classes, properties, and relations between the data, and entities that substantiate considered domains.

The Resource Description Framework (RDF) provides a graph-based data model or framework for structuring data as statements about resources. A "resource" may be any "thing" that exists in the world: a person, place, event, book, museum object, but also an abstract concept like data objects. Figure 1.6 shows an RDF graph.

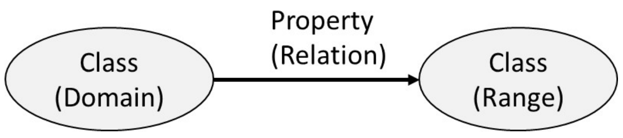

Figure 1.6: RDF graphic

RDF is based on the idea of making statements about resources (in particular web resources) in expressions of the form subject–predicate–object, known as triples. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. In the context of ontology, the term subject expresses what the sentence is about (topic) (see 1.1).

For describing ontologies several languages have been developed. One widely used language is OWL (worldwide web ontology language) which is based on the Resource Description Framework (RDF).

OWL has classes, properties, and instances. Classes represent terms also called concepts. Classes have properties and instances are individuals of one or more classes.

A class is a type of thing. A type of "resource" in the RDF sense can be person, place, object, concept, event, etc.. Classes and subclasses form a hierarchical taxonomy and members of a subclass inherit the characteristics of their parent class (superclass). Everything true for the parent class is also true for the subclass.

A member of a subclass "is a", or "is a kind of" its parent class. Ontologies define a set of properties used in a specific knowledge domain. In an ontology context, properties relate members of one class to members of another class or a literal.

Domains and ranges define restrictions on properties. A domain restricts what kinds of resources or members of a class can be the subject of a given property in an RDF triple. A range restricts what kinds of resources/members of a class or data types (literals) can be the object of a given property in an RDF triple.

Entities belonging to a certain class are instances of this class or individuals. A simple ontology with various classes, properties and individual is shown below:

Ontology statement examples:

- **Class definition statements:**
  - Parent `isA` Class
  - Mother `isA` Class
  - Mother `subClassOf` Parent
  - Child `isA` Class
- **Property definition statement:**
  - `isMotherOf` is a relation between the classes Mother and Child
- **Individual/instance statements:**
  - MariaSchmidt `isA` Mother
  - MaxSchmidt `isA` Child
  - MariaSchmidt `isMotherOf` MaxSchmidt

1.3 Introduction to Abstract State Machines

An abstract state machine (ASM) is a state machine operating on states that are arbitrary data structures (structure in the sense of mathematical logic, that is a nonempty set together with several functions (operations) and relations over the set).

The language of the so-called Abstract State Machine uses only elementary If-Then-Else-rules which are typical also for rule systems formulated in natural language, i.e., rules of the (symbolic) form

if *Condition* **then** *ACTION*

with arbitrary *Condition* and *ACTION*. The latter is usually a finite set of assignments of the form *f (t1, ..., tn) := t*. The meaning of such a rule is to perform in any given state the indicated action if the indicated condition holds in this state.

The unrestricted generality of the used notion of Condition and *ACTION* is guaranteed by using as ASM-states the so-called Tarski structures, i.e., arbitrary sets of arbitrary elements with arbitrary functions and relations defined on them. These structures are updatable by rules of the form above. In the case of business processes, the elements are placeholders for values of arbitrary type and the operations are typically the creation, duplication, deletion, or manipulation (value change) of objects. The so-called views are conceptually nothing else than projections (read: substructures) of such Tarski structures.

An (asynchronous, also called distributed) ASM consists of a set of agents each of which is equipped with a set of rules of the above form, called its program. Every agent can execute in an arbitrary state in one step all its executable rules, i.e., whose condition is true in the indicated state. For this reason, such an ASM, if it has only one agent, is also called sequential ASM. In general, each agent has its own "time" to execute a step, in particular, if its step is independent of the steps of other agents; in special cases, multiple agents can also execute their steps simultaneously (in a synchronous manner).

Without further explanations, we adopt usual notations, abbreviations, etc., for example:

```
if Cond then M1 else M2
```

instead of the equivalent ASM with two rules:

```
if Cond then M1
if not Cond then M2
```

Another notation used below is

```
let x=t in M
```

for $M(x/a)$, where a denotes the value of t in the given state and $M(x/a)$ is obtained from M by substitution of each (free) occurrence of x in M by a.

For details of a mathematical definition of the semantics of ASMs which justifies their intuitive (rule-based or pseudo-code) understanding, we refer the reader to the AsmBook Börger, E., Stärk R. Abstract State Machines. A Method for High-Level System Design and Analysis. Springer, 2003.

CHAPTER 2

Structure of a PASS Description

In this chapter we describe the structure of a PASS specification. The structure of a PASS description consists of the subjects and the messages they exchange.

2.1 Informal Description

2.1.1 Subject

Subjects represent the behavior of an active entity. A specification of a subject does not say anything about the technology used to execute the described behavior. Subjects communicate with each other by exchanging messages. Messages have a name and a payload. The name should express the meaning of a message informally and the payloads are the data (business objects) transported. Internal subjects execute local activities such as calculating a price, storing an address etc. External subjects represent interfaces for other business processes.

A subject sends messages to other subjects, receives messages from other subjects, and executes internal actions. All these activities are done in logical order which is defined in a subject's behavior specification.

In the following we use an example for the informal definition of subjects. In the simple scenario of the business trip application, we can identify three subjects, namely the employee as applicant, the manager as the approver, and the travel office as the travel arranger.

In general, there are the following types of subjects:

- Fully specified subjects
- Multi-subjects
- Single subjects
- Interface subjects

Fully specified Subjects

This is the standard subject type. A subject communicates with other subjects by exchanging messages. Fully specified subjects consists of following components:

- Business Objects—Each subject has some business objects. A basic structure of business objects consists of an identifier, data structures, and data elements. The identifier of a business object is derived from the business environment in which it is used. Examples are business trip requests, purchase orders, packing lists, invoices, etc. Business objects are composed of data structures. Their components can be simple data elements of a certain type (e.g., string or number) or even data structures themselves.

- Sent messages—Messages which a subject sends to other subjects. Each message has a name and may transport some data objects as a payload. The values of these payload data objects are copied from internal business objects of a subject.

- Received messages—Messages received by a subject. The values of the payload objects are copied to business objects of the receiving subject.

- Input Pool—Messages sent to a subjects are deposited in the input pool of the receiving subject. The input pool is a very important organizational and technical concept in this case.

- Behavior—The behavior of each subject describes in which logical order it sends messages, expects (receives) messages, and performs internal functions. Messages transport data from the sending to the receiving subject, and internal functions operate on internal data of a subject.

Multsubjects and Multiprocesses

Multi-subjects are similar to fully specified subjects. If in a process model several identical subjects are required, e.g. in order to increase the throughput, this requirement can be modeled by a multi-subject. If several communicating subjects in a process model are multi-subjects they can be combined to a multi-process.

In a business process there may be several identical sub-processes that perform certain similar tasks in parallel and independently. This is often the case in a procurement process, when bids from multiple suppliers are solicited. A process or sub-process is therefore executed simultaneously or sequentially multiple times during overall process execution. A set of type-identical, independently running processes or sub-processes are termed multi-process. The actual number of these independent sub-processes is determined at runtime.

Multi-processes simplify process execution, since a specific sequence of actions can be used by different processes. They are recommended for recurring structures and similar process flows.

An example of a multi-process can be illustrated as a variation of the current booking process. The travel agent should simultaneously solicit up to five bids before making a reservation. Once three offers have been received, one is selected and a room is booked. The process of obtaining offers from the hotels is identical for each hotel and is therefore modeled as a multi-process.

Single subjects

Single subjects can be instantiated only once. They are used if for the execution of a subject a resource is required which is only available once.

Interface Subjects

Interface subjects are used as interfaces to other process systems. If a subject of a process system sends or receives messages from a subject which belongs to an other process system. These so called interface subjects represent fully described subjects which belong to that other process system. Interface subjects specifications contain the sent messages, received messages and the reference to the fully described subject which they represent.

2.1.2 Subject-to-Subject Communication

After the identification of subjects involved in the process (as process-specific roles), their interaction relationships need to be represented. These are the messages exchanged between the subjects. Such messages might contain structured information—so-called business objects.

The result is a model of the communication relationships between two or more subjects, which is referred to as a **Subject Interaction Diagram** (SID) or, synonymously, as a Communication Structure Diagram (CSD) (see figure 2.1).

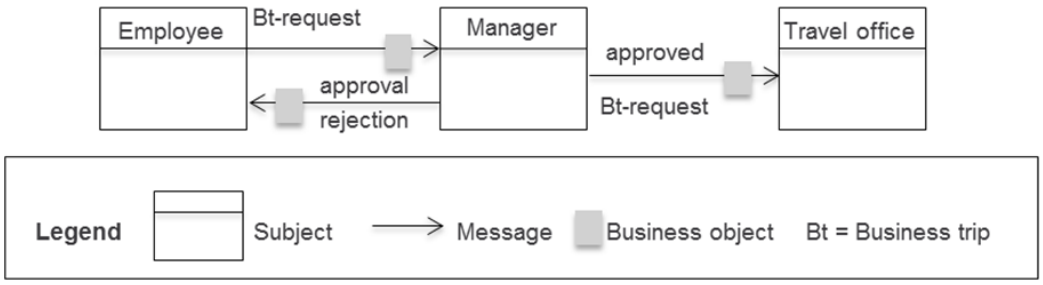

Figure 2.1: Subject interaction diagram for the process ’business trip application’

Messages represent the interactions of the subjects during the execution of the process. We recommend naming these messages in such a way that they can be immediately understood and also reflect the meaning of each particular message for the process. In the sample ’business trip application’, therefore, the messages are referred to as ’business trip request’, ’rejection’, and ’approval’.

Messages serve as a container for the information transmitted from a sending to a receiving subject. There are two options for the message content:

- Simple data types—Simple data types are string, integer, character, etc. In the business trip application example, the message ’business trip request’ can contain several data elements of type string (e.g., destination, reason for traveling, etc.), and of type number (e.g., duration of trip in days).

- Business Objects—Business Objects in their general form are physical and logical ’things’ that are required to process business transactions. We consider data structures composed of elementary data types, or even other data structures, as logical business objects in business processes. For instance, the business object ’business trip request’ could consist of the data structures ’data on applicants’, ’travel data’, and ’approval data’ with each of these in turn containing multiple data elements.

2.1.3 Message Exchange

In the previous subsection, we have stated that messages are transferred between subjects and have described the nature of these messages. What is still missing is a detailed description of how messages can be exchanged, how the information they carry can be transmitted, and how subjects can be synchronized. These issues are addressed in the following sub-sections.

Synchronous and Asynchronous Exchange of Messages

In the case of synchronous exchange of messages, sender and receiver wait for each other until a message can be passed on. If a subject wants to send a message and the receiver (subject) is not yet in a corresponding receive state, the sender waits until the receiver is able to accept this message. Conversely, a recipient has to wait for a desired message until it is made available by the sender.

The disadvantage of the synchronous method is a close temporal coupling between sender and receiver. This raises problems in the implementation of business processes in the form of workflows, especially across organizational borders. As a rule, these also represent system boundaries across which a tight coupling between sender and receiver is usually very costly. For long-running processes, sender and receiver may wait for days, or even weeks, for each other.

Using asynchronous messaging, a sender is able to send anytime. The subject puts a message into a message buffer from which it is picked up by the receiver. However, the recipient sees, for example, only the oldest message in the buffer (in case the buffer is implemented as FIFO or LIFO storage) and can only accept this particular one. If it is not the desired message, the receiver is blocked, even though the message may already be in the buffer, but in a buffer space that is not visible to the receiver. To avoid this, the recipient has the alternative to take all of the messages from the buffer and manage them by himself. In this way, the receiver can identify the appropriate message and process it as soon as he or she needs it. In asynchronous messaging, sender and receiver are only loosely coupled. Practical problems can arise due to the in reality limited physical size of the receive buffer, which does not allow an unlimited number of messages to be recorded. Once the physical boundary of the buffer has been reached due to high occupancy, this may lead to unpredictable behavior of workflows derived from a business process specification. To avoid this, the input-pool concept has been introduced in PASS. Nevertheless, the number of messages must always be limited, as a business process must have the capacity to handle all messages to maintain some sort of service level.

Exchange of Messages via the Input Pool

To solve the problems outlined in asynchronous message exchange, the input pool concept has been developed. Communication via the input pool is considerably more complex than previously shown; however, it allows transmitting an unlimited number of messages simultaneously. Due to its high practical importance, it is considered as a basic construct of PASS.

Consider the input pool as a mail box of work performers, the operation of which is specified in detail. Each subject has its own input pool. It serves as a message buffer to temporarily store messages received by the subject, independent of the sending communication partner. The input pools are therefore inboxes for flexible configuration of the message exchange between the subjects.

In contrast to the buffer in which only the front message can be seen and accepted, the pool solution enables picking up (i.e. removing from the buffer) any message. For a subject, all messages in its input pool are visible.

The input pool has the following configuration parameters (see figure 2.2):

- Input-pool size—The input-pool size specifies how many messages can be stored in an input pool, regardless of the number and complexity of the message parameters transmitted with a message. If the input pool size is set to zero, messages can only be exchanged synchronously.

- Maximum number of messages from specific subjects—For an input pool, it can be determined how many messages received from a particular subject may be stored simultaneously in the input pool. Again, a value of zero means that messages can only be accepted synchronously.

- Maximum number of messages with specific identifiers—For an input pool, it can be determined how many messages of a specifically identified message type (e.g., invoice) may be stored simultaneously in the input pool, regardless of what subject they originate from. A specified size of zero allows only for synchronous message reception.

- Maximum number of messages with specific identifiers of certain subjects—For an input pool, it can be determined how many messages of a specific identifier of a particular subject may be stored simultaneously in the input pool. The meaning of the zero value is analogous to the other cases.

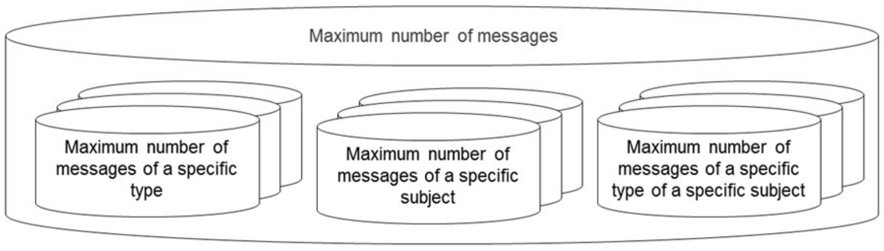
Maximum number of messages
Maximum number of messages of a specific type
Maximum number of messages of a specific subject
Maximum number of messages of a specific type of a specific subject

Figure 2.2: Configuration of Input Pool Parameters

By limiting the size of the input pool, its ability to store messages may be blocked at a certain point in time during process runtime. Hence, messaging synchronization mechanisms need to control the assignment of messages to the input pool. Essentially, there are three strategies to handle the access to input pools:

- Blocking the sender until the input pool's ability to store messages has been reinstated—Once all slots are occupied in an input pool, the sender is blocked until the receiving subject picks up a message (i.e. a message is removed from the input pool). This creates space for a new message. In case several subjects want to put a message into a fully occupied input pool, the subject that has been waiting longest for an empty slot is allowed to send. The procedure is analogous if corresponding input pool parameters do not allow storing the message in the input pool, i.e., if the corresponding number of messages of the same name or from the same subject has been put into the input pool.

- Delete and release of the oldest message—In case all the slots are already occupied in the input pool of the subject addressed, the oldest message is overwritten with the new message.
- Delete and release of the latest message—The latest message is deleted from the input pool to allow depositing of the newly incoming message. If all the positions in the input pool of the addressed subject are taken, the latest message in the input pool is overwritten with the new message. This strategy applies analogously when the maximum number of messages in the input pool has been reached, either with respect to sender or message type.

2.2 OWL Description

The various building blocks of a PASS description and their relations are defined in an ontology. The following figure 2.3 gives an overview of the structure of the PASS specifications.

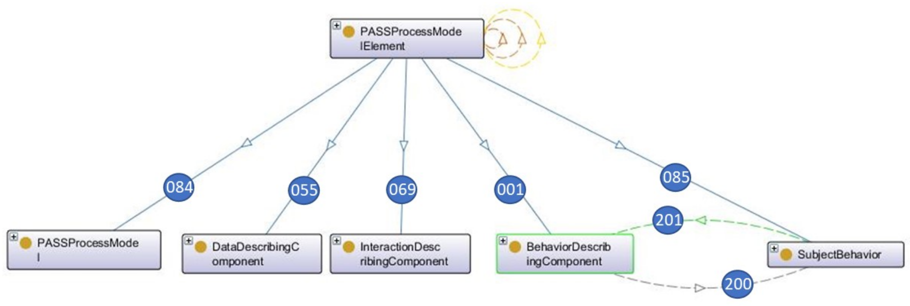

Figure 2.3: Elements of PASS Process Models

The class `PASSProcessModelElement` has five subclasses (subclass relations 084, 055, 069, 001 and 085 in figure 2.3). Only the classes `PASSProcessModel`, `DataDescriptionCOmponent`, `InteractionDescribingComponent` are used for defining the structural aspects of a process specification in PASS. The classes `BehaviorDescribingComponent` and `SubjectBehavior` define the dynamic aspects. In which sequences messages are sent and received or internal actions are executed. These dynamic aspects are considered in detail in the next chapter.

2.2.1 PASS Process Model

The central entities of a PASS process model are subjects which represents the active elements of a process and the messages they exchange. Messages transport data from one subject to others (payload). Figure 2.4 shows the corresponding ontology for the PASS process models.

`PASSProcessModelElements` and `PASSProcessModells` have a name. This is described with the property `hasAdditionalAttribute` (property 208 in 2.3). The class subject and the class `MessageExchange` have the relation `hasRelation toModelComponent` to the class `PASSProcessModel` (property 226 in 2.3). The

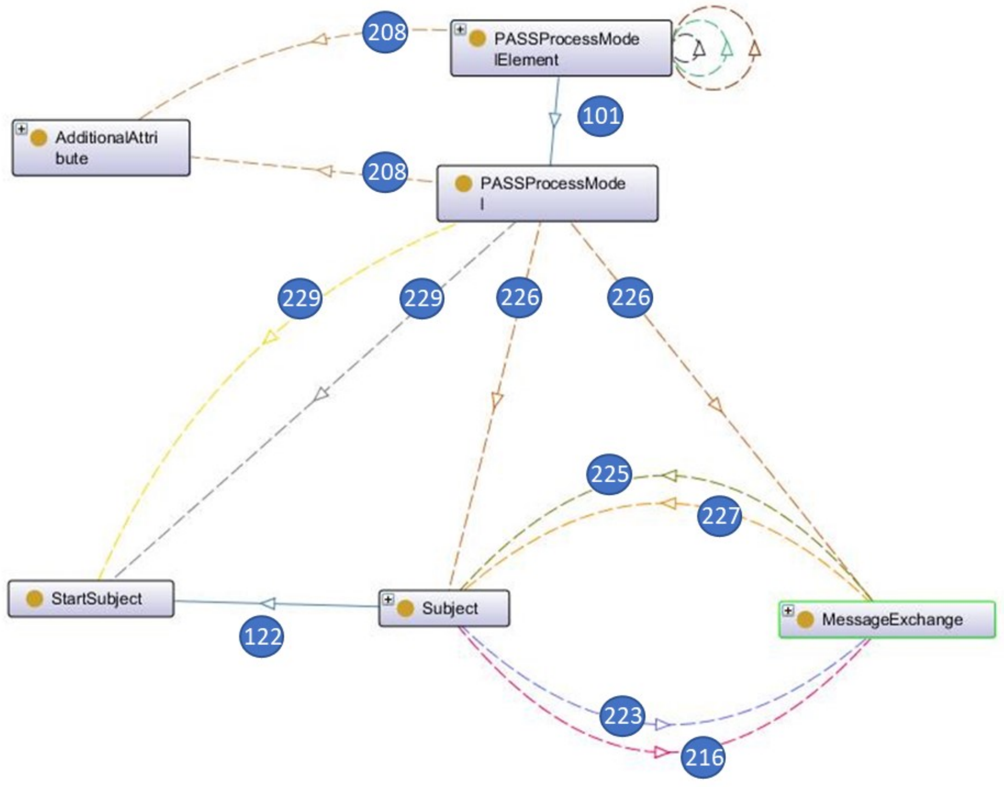

Figure 2.4: PASS Process Modell

properties `hasReceiver` and `hasSender` express that a message has a sending and receiving subject (properties 225 and 227 in 2.3) whereas the properties `hasOutgoingMessageExchange` and `hasIncomingMessageExchange` define which messages are sent or received by a subject. Property `hasStartSubject` (property 229 in 2.3) defines a start subject for a `PASSProcessModell`. A start subject is a subclass of the class subject (subclass relation 122 in 2.3).

2.2.2 Data Describing Component

Each subject encapsulate data (business objects). The values of these data elements can be transferred to other subjects. The following figure 2.5 shows the ontology of this part of the PASS-ontology.

Three subclasses are derived from the class `DtadescribingCombonent` (in figure 2.5 are these the relations 060, 056 and 066). The subclass `PayLoadDescription` defines the data tranported by messages. The relation of `PayloadDescriptions` to messages is defined by the property `ContainsPayloadDescription` (in figure 2.5 number 204).

There are two types of payloads. The class `PayloadPhysicalObjectDescription` is used if a message will be later implemented by a physical transport like a parcel. The class `PayLoadDataObjectDefinition` is used to transport normal data (Subclass relations 068 and 67 in figure 2.5). These payload objects are also a subclass of the class `DataObjectDefinition` (Subclass relation 058 in figure 2.5).

Data objects have a certain type. Therefore class `DataObjectDefinition` has the relation `hasDatatype` to class `DataTypeDefinition` (property 212 in

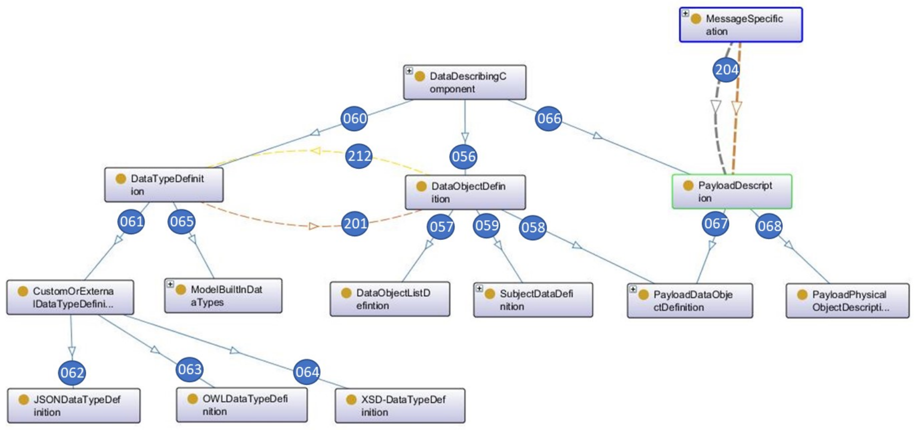

Figure 2.5: Data Description Component

figure2.5). Class DataTypeDefinition has two subclasses (subclass relations 061 and 065 in figure 2.5). The subclass ModelBuiltInDataTypes are user defined data types whereas the class CustomOfExternalDataTypeDefinition is the superclass of JSON, OWL or XML based data type definitions(subclass relations 062, 63 and 064 in figure 2.5).

2.2.3 Interaction Describing Component

The following figure 2.6 shows the subset of the classes and properties required for describing the interaction of subjects.

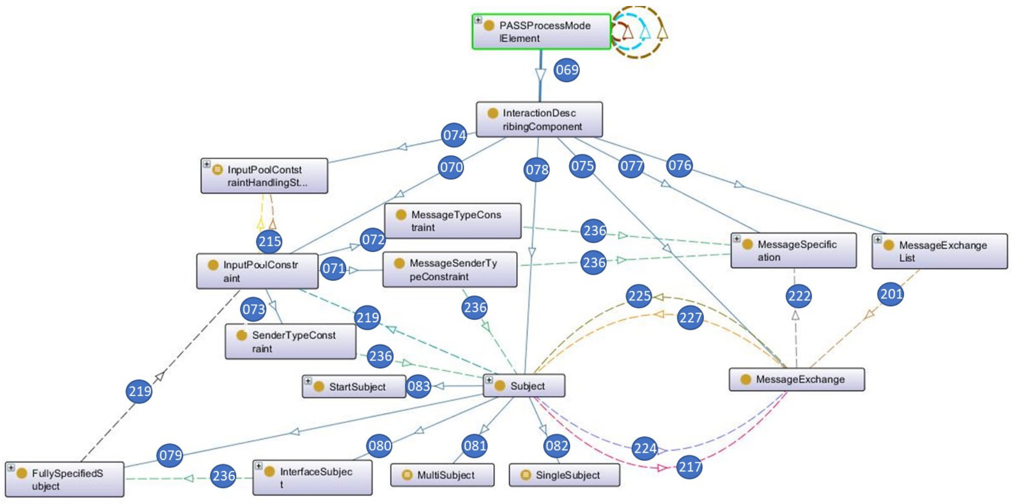

Figure 2.6: Subject Interaction Diagram

The central classes are Subject and MessageExchange. Between these classes are defined the properties hasIncomingTransition (in figure 2.6 number 217) and hasOutgoingTransition (in figure 2.6 number 224). This properties defines that subjects have incoming and outgoing messages. Each message has a sender and a receiver (in figure 2.6 number 227 and number 225). Messages have a type. This is expressed by the property hasMessageType (in figure 2.6

number 222). Instead of the property 222 a message exchange may have the property 201 if a list of messages is used instead of a single message.

Each subject has an input pool. Input pools have three types of constraints (see section 2.1.3). This is expressed by the property references (in figure 2.6 number 236) and `InputPoolConstraints` (in figure 2.6 number 219). Constraints which are related to certain messages have references to the class `MessageSpecification`.

There are four subclasses of the class `subject` (in figure 2.6 number 079, 080, 081 and 082). The specialties of these subclasses are described in section 2.1.1. A class `StartSubject` (in figure 2.6 number 83) which is a subclass of class subject denotes the subject in which a process instance is started.

All other relations are subclass relations. The class `PASSProcessModelElement` is the central PASS class. From this class all the other classes are derived (see next sections). From class `InteractionDescribingCOmponent` all the classes required for describing the structure of a process system are derived.

2.3 ASM Description

In this chapter only the structure of a PASS model is considered. Execution has not been considered. Because ASM only considers execution aspects in this chapter an ASM specification of the structural aspects does not make sense. The execution semantic is part of chapter 4.

Bibliography

[Kee76] E. L. Keenan. 1976.