S-BPM Method by Comparison 14

14.1 To Go

I have noted with dismay that we indeed (still) have our existing data and functional models of work organization. What should we do with these?

Alignment should be easy, since S-BPM covers both structural and functional characteristics, and enriches these with the actor's perspective. This ultimately facilitates the embedding of existing specifications, such as data models, into business objects, so that all of the structures can again be found in S-BPM-models.

I suggest that we look at each model and existing specifications, and adjust them first with the respective S-BPM dimensions. In addition to the existing structural descriptions, such as data models, which can be attributed to the object perspective, we will work on the functional processes including the flow of information and communication, in order to identify subjects and predicates. In this way, we can collect all of the ingredients for integration.

The basis is the behavior as described in my subject diagram.

This book provides comprehensive insights into the subject-oriented methodology. In addition to deriving and justifying the concept, we have developed a subject-oriented process model for dealing with models. To complete the picture with respect to BPM, we examine the extent to which other methods also comprise subject-oriented elements. The focus on subjects while reflecting standard sentence semantics of natural language can be spotted in the canon of existing approaches for modeling business processes in various places. The following overview of essential diagrammatic or formal modeling methods for business processes shows the different links of existing approaches to the modeling categories subject, predicate, and object. The respective approaches are comparatively described.

After a review of the concepts for modeling, we follow the historical development of business process modeling and start with activity- or function-oriented approaches—they refer to the predicate. The object-oriented approaches stem from software engineering and refer to objects. The subject reference can be traced back to the theory of process-directed data processing. Finally, there are integrated approaches that include at least two of the three constituent characteristics of subject-oriented business process modeling.

14.2 Subject, Predicate, and Object in Modeling

Business processes are sequences of actions in a company that will be described by a model. Developing business processes means that a model of the existing or a new requirement for a target business process is created.

Business processes can also be interpreted as descriptions of socio-technical systems (Sinz 2010). Business process models describe the properties and behavior of process participants and their interaction with(in) the technical and organizational environment. These models can be viewed from different perspectives. The process of model construction is preceded by an analysis that leads to specific facts either being considered essential or merely supplemental (cf. Scholz and Holl 1999; Denert 1991). In Scholz and Holl (1999), crucial model elements are termed essentials and complimentary ones accidentals.

Depending on which model elements are considered essential when defining business processes, different approaches to modeling are used. Accidental elements are grouped around essential ones. The following aspects of modeling are currently being used (cf. Scholz and Holl 1999; Denert 1991):

- The functional approach focuses on functions. Examples of function-oriented models are control flow diagrams and data flow diagrams according to de-Marco (1979) or Event-driven Process Chains (EPCs).
- In data-driven approaches, accidents are grouped around data. A well-known example of data-driven modeling approaches is Entity-Relationship Diagrams.
- In the object-oriented approach, accidents are grouped around objects. Objects in computer science are data structures, encapsulated with the operations on these data structures. The object-oriented modeling approach is currently considered the most accepted. A well-known method of description is the Unified Modeling Language (UML).

A prerequisite for modeling is that the models are adequately described and documented, so that they can be understood by all and model content can be communicated or discussed. Models are used in particular in BPM for analysis of business processes with the involvement of different actors.

In the above list, some well-known languages for documenting results of process analysis have been given. Modeling, ultimately, describes part of reality using an "artificial" language. A model is thus an artifact, an artificially created structure which contains an excerpt of the reality as perceived by humans. The formalism of models for business processes is such that they can be mapped to IT. In the last few decades in computer science, a paradigm shift from flow orientation to object orientation has occurred. Applied to modeling, the essential aspects have been shifted from the predicate (batch processing, while . . .do. . .) to the object, while subjects were treated only rudimentarily so far. Subject-oriented business process modeling puts the subject into the center of attention. Participants of the S-BPM ONE 2010 congress in Karlsruhe created the hypothesis that after 1970 and 1990, the year 2010 could mark the beginning of a new paradigm switch, namely to subject orientation (see Fig. 14.1).

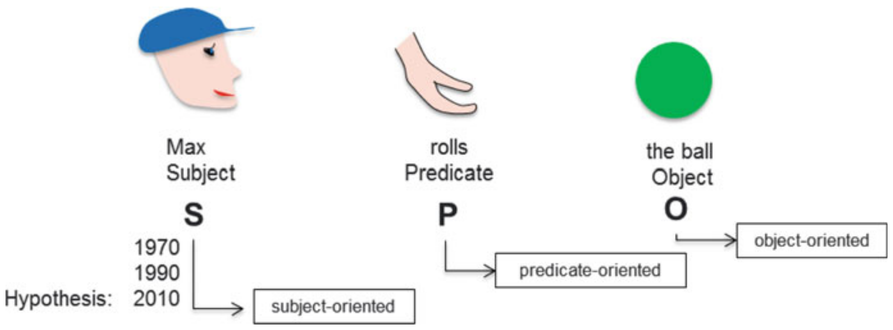


Fig. 14.1 Temporal evolution of flow orientation, object orientation, and subject orientation

14.3 Comparative Analysis

In the following, the best-known modeling approaches are presented and analyzed for their coverage of the natural language sentence semantics and the resulting impact for modeling. Finally, these are compared with the subject-oriented modeling approach.

We exemplify the different approaches using the process for applying for a business trip. It will be shown, in which models generally available for practical description and definition of application programs in computer science, which parts of the standard semantics of subject–predicate–object correspond to essential or accidental elements, and how the process can be described in the respective modeling approach.

We start out with the natural language description of the business trip application process (see Fig. 14.2). This description focuses on the elements perceived as essential aspects of the process when applying for business trips. It will now be specified using various formal or semiformal modeling methods. The relevant sections provide a brief overview of the history of the respective category of approaches, before explaining their representatives in an exemplary way.

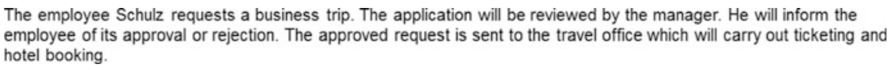

The employee Schulz requests a business trip. The application will be reviewed by the manager. He will inform the employee of its approval or rejection. The approved request is sent to the travel office which will carry out ticketing and hotel booking.

Fig. 14.2 Natural language description of the business trip application process

14.3.1 Modeling While Focusing on Predicates

14.3.1.1 Origin

In the beginning of data processing in the 1970s, mechanical and automated processing was at the forefront. In mainframe data processing, actions were at the center of attention. Terms such as “operator” or “data or information processing” were coined at that time. Even in the first programming languages, operational constructs are in the foreground; their core consists of commands such as “while . . . do . . .”. The first computer systems were built to solve complex computational problems of the time, stemming from mathematics or physics. For instance, the trained civil engineer Konrad Zuse wanted to automate his statics’ calculations and built the first calculating machine. For these activities, calculations were at the focus of attention. The data were parameters of mathematical or physical formulas and played a secondary role. Likewise, the actor, or the subject, was of minor importance. The subject was the person interested in the results of the calculation. The focus was on the action, i.e., the predicate. Programming was meant to define complex sequences of actions.

14.3.1.2 Flowcharts

One of the first models for algorithmic tasks was flowcharts or program flowcharts. Flowcharts describe a sequence of operations to solve a task. A business trip application can be mapped to a flowchart (see Fig. 14.3).

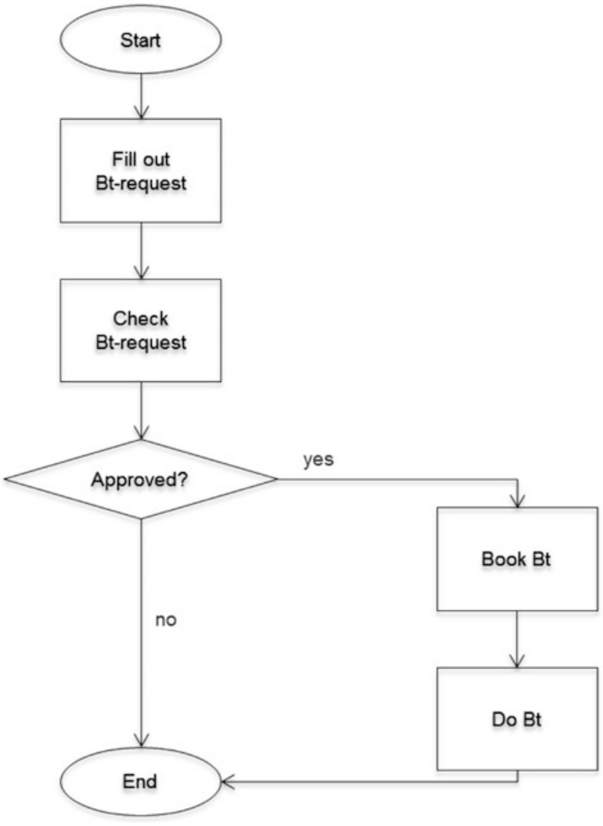


Fig. 14.3 Business trip application process as a flowchart

When flowcharts are used to describe a computational algorithm, it is clear who initiates the individual actions in the flowchart: it is the person carrying out the task, or the executing computer system. These standard subjects are not mentioned explicitly. In addition, the data required for executing a flowchart are specified only rudimentarily.

Using flowcharts, natural language supplements, such as subjects and objects, can be added, but they are not integrated in the logic of the model. Figure 14.4 shows the example extended to subjects. They were added in natural language.

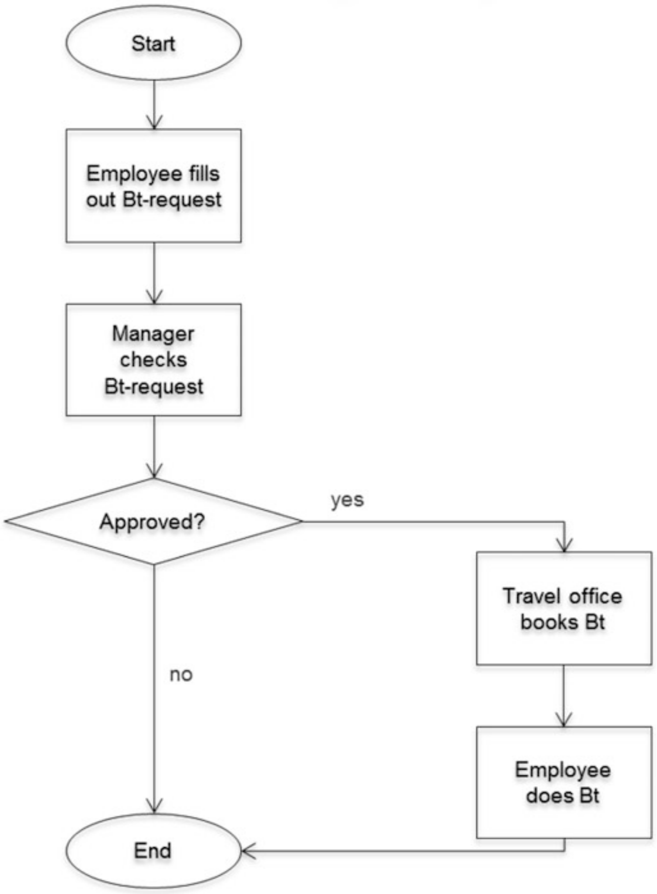


Fig. 14.4 Business trip application process as a flowchart including subjects

In advanced forms of flowcharts, in addition to the verbs, the subjects and objects are directly or indirectly represented as symbols. Figure 14.5 shows the previous flowchart after adding the subjects “employee” and “manager” indirectly by adding the symbols for the manual entry of the business trip application and the decision-making results. The modified diagram also contains an object represented by the symbol for a data set (business trip data).

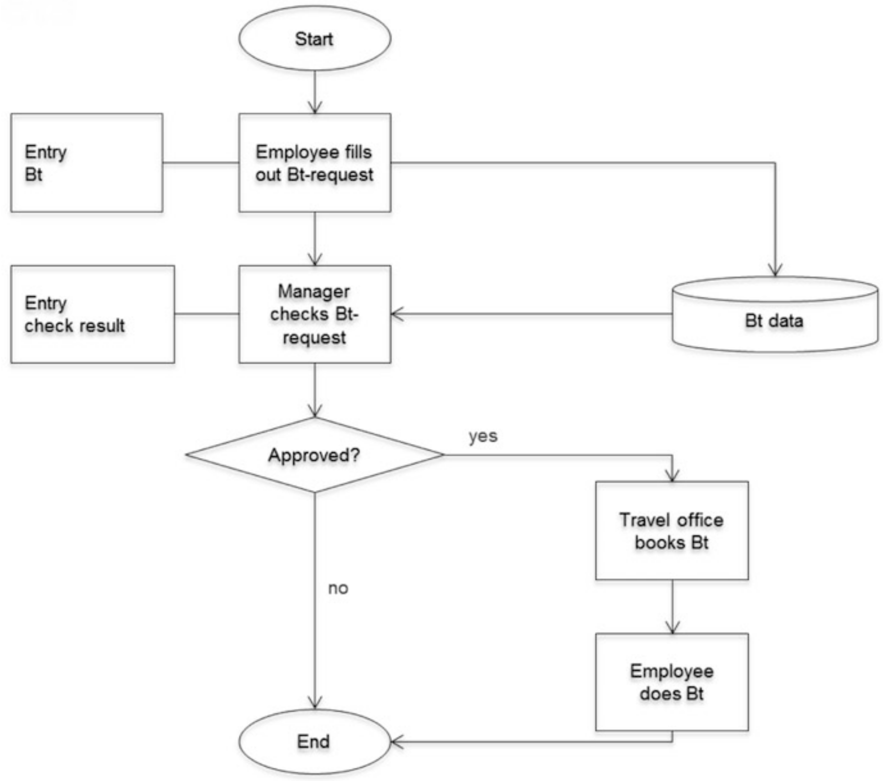


Fig. 14.5 Business trip application process as a flowchart including subjects and objects

14.3.1.3 Event-Driven Process Chains

A control-flow-based method for representing business processes is Event-driven Process Chains (EPC). Figure 14.6 shows the process of the business trip application as an EPC.

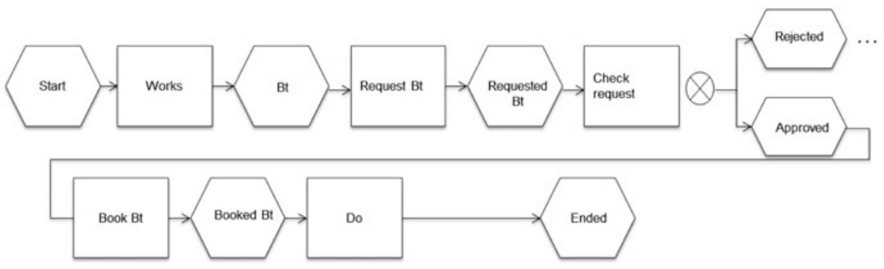


Fig. 14.6 Business trip application process as an EPC

The rectangles represent the actions of a process that may contain natural language objects for illustration purposes. The individual actions are preceded by an event (hexagons), which represents the impulse to perform an action or the result of the previous action. With the help of connectors, the results of a function can lead to different events. The action “check request” could either lead to the event “rejected” or “approved” (XOR). In addition to XOR, there are other connectors. Details of EPCs and their use are described in Scheer (1998).

In practice, today mainly extended EPCs (eEPCs) are used. These complement the original EPCs with elements of organization, data, and performance modeling. These amendments correspond essentially to subjects and objects.

Figure 14.7 shows an extended EPC of the business trip application process.

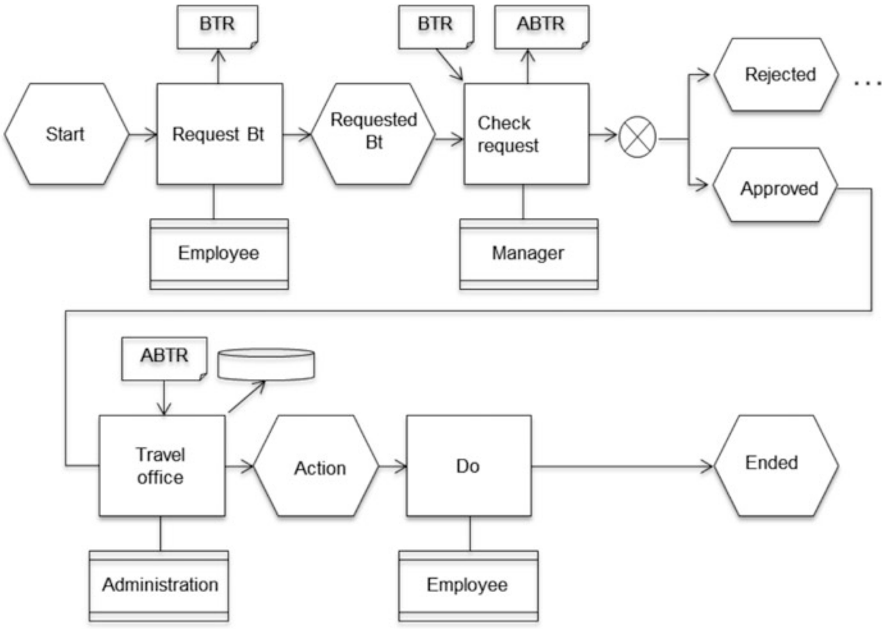


Fig. 14.7 Business trip application process as eEPC including subject, predicate, and object

Hereby, eEPCs in principle allow representing all language constructs. In such a representation, functions are still at the center of attention. An identification of the subject including its entire behavior is not possible due to the distributed representation of the subject in the diagram.

14.3.1.4 Petri Nets

An important model in theoretical computer science is Petri nets (cf. Stucky and Winand 1997). They are an action-oriented modeling method, i.e., Petri nets are predicate oriented. In contrast to control flow diagrams, they allow performing multiple actions in parallel.

In order to also support data aspects, attributed Petri nets have been developed. However, approaches to represent subjects are still missing.

Figure 14.8 shows a Petri net for the business trip application process. A Petri net consists of an initial marking, places (solid bars), transitions (ovals), and arcs (arrowed lines). Arcs connect transitions to places or places to transitions, but never places to places or transitions to transitions. In general, transitions are interpreted as actions and places as conditions for a transition. A transition can switch when in its input places there is at least one so-called token. After switching, each output place receives a token. The initial marking determines which places have tokens to start the execution. In the figure, the place "employee requests business trip" contains the token.

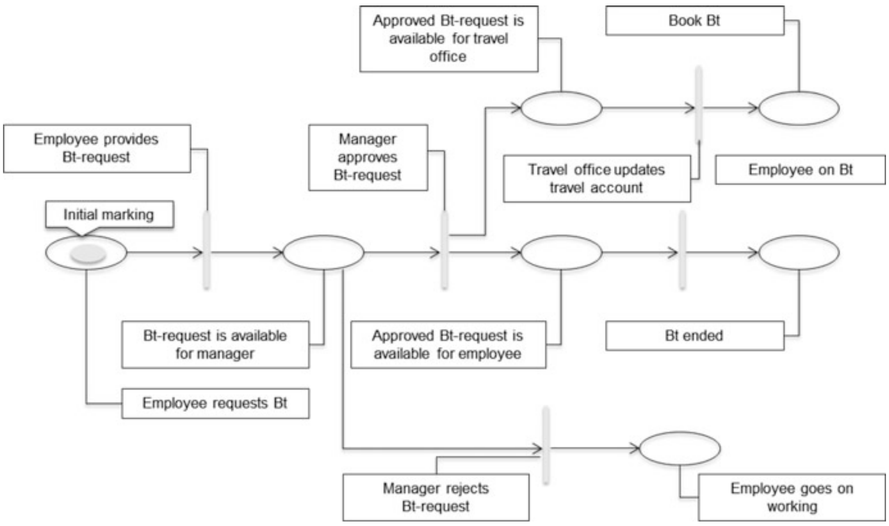

Approved Bt-request is available for travel office
Book Bt
Employee provides Bt-request
Manager approves Bt-request
Travel office updates travel account
Employee on Bt
Initial marking
Bt-request is available for manager
Approved Bt-request is available for employee
Bt ended
Employee requests Bt
Manager rejects Bt-request
Employee goes on working

Fig. 14.8 Business trip application process as a Petri net with initial marking

After switching the transition "employee provides business trip request", the token is reassigned as shown in Fig. 14.9. The token is removed from the place "employee requests business trip" and a token appears in the place "business trip request is available for manager".

After that, either the transition "manager rejects business trip request" or the transition "manager approves business trip request" can switch. The Petri net is therefore referred to as nondeterministic. In case the transition "manager approves business trip request" switches, the places "approved business trip request is available for travel office" and "approved business trip request is available for employee" are each provided with a token (see Fig. 14.10).

The example reveals that Petri nets focus on the sequence of actions. Subjects and objects are complemented by natural language comments. In this case, this is done by selecting appropriate names for the places and transitions. The advantage of Petri nets as compared to flowcharts is that they are grounded in theory and concurrency can be represented.

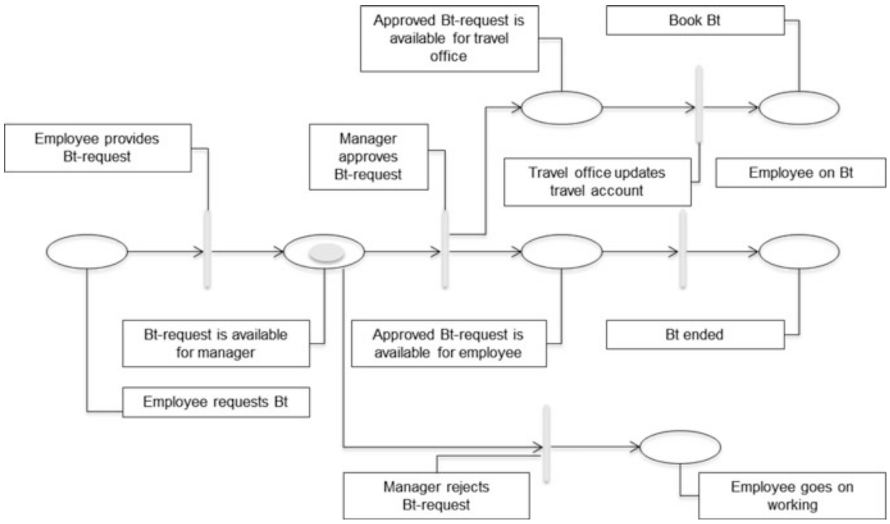

Approved Bt-request is available for travel office
Book Bt
Employee provides Bt-request
Manager approves Bt-request
Travel office updates travel account
Employee on Bt
Bt-request is available for manager
Approved Bt-request is available for employee
Bt ended
Employee requests Bt
Manager rejects Bt-request
Employee goes on working

Fig. 14.9 Business trip application process as a Petri net with tokens assignment after switching "employee provides business trip request"

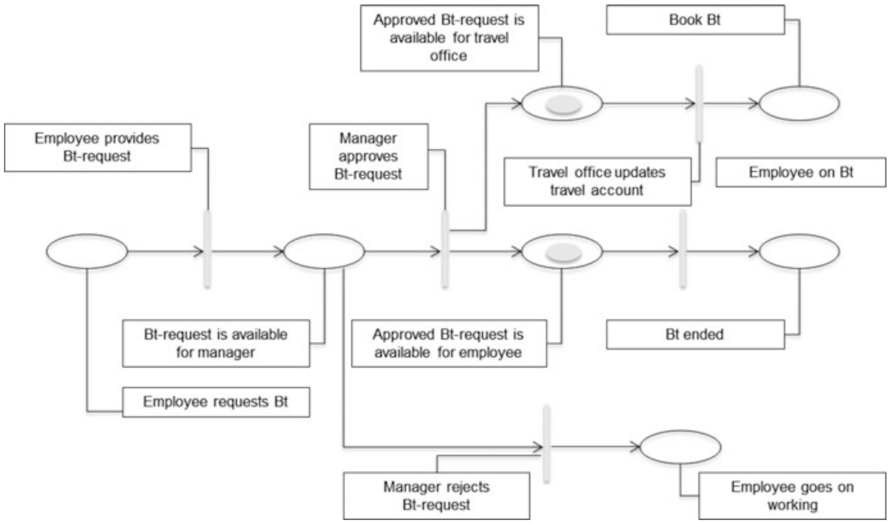

Approved Bt-request is available for travel office
Book Bt
Employee provides Bt-request
Manager approves Bt-request
Travel office updates travel account
Employee on Bt
Bt-request is available for manager
Approved Bt-request is available for employee
Bt ended
Employee requests Bt
Manager rejects Bt-request
Employee goes on working

Fig. 14.10 Business trip application process as a Petri net with tokens after switching "manager approves business trip request"

14.3.2 Modeling While Focusing on Objects

14.3.2.1 Origin

With the increasing use of computer systems in industry, the aspect of data management and data processing has become increasingly important. In companies, large data sets, such as order or invoice data, need to be stored and manipulated. To meet these requirements, modeling languages have been developed which bring the target of actions, namely the objects or data, to the focus of attention.

14.3.2.2 Entity-Relationship Model

The Entity-Relationship Model (ER Model or ERM) describes data entities and their mutual relationships. ER models are usually represented graphically. Their advantage is their ability to map complex worlds using simple tools:

- Entity: object of actual world, either material or abstract (e.g., employee "Schulz", manager "Schmid").
- Relationship: semantic relationship between two or more objects (e.g., employee "Schulz" "is a staff member" of manager "Schmid").
  The model itself consists exclusively of entity types and relationship types:
- Entity type: typifying of similar entities (e.g., employee and manager), shown as a rectangle.
- Relationship type: typifying of similar relationships (e.g., "is employee of"). The semantics of the relationship between entity types is expressed in the ER diagram by a short text label on the border, while it is left up to the modeler what name he provides.

Figure 14.11 shows the ERM of the business trip application process. Each employee has exactly one manager and each manager is boss of 1 to *n* employees. Each employee has applied for none or up to *n* business trips. Each business trip request contains exactly one travel date for the beginning and the end of the business trip, respectively. A manager has to decide upon 0 to *m* business trip requests.

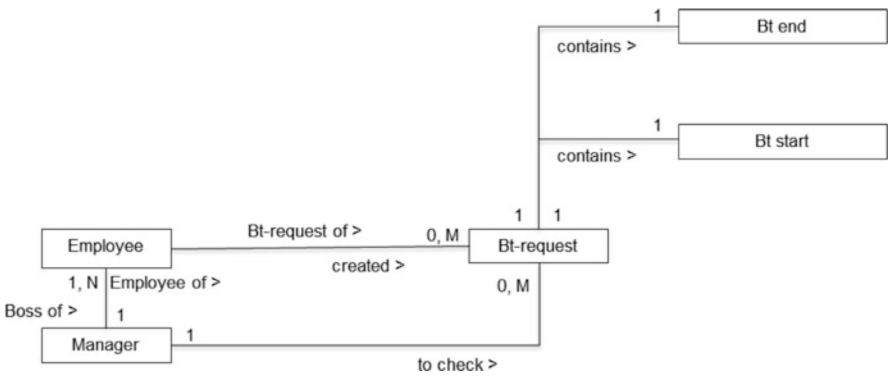


Fig. 14.11 ERM for the business trip application process

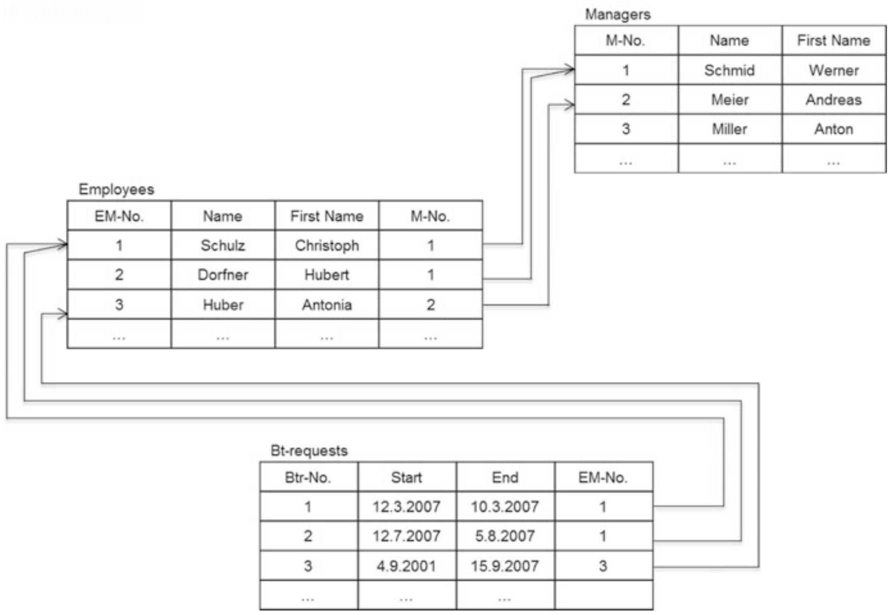
An ERM is focused on objects. Subjects and predicates are only indirectly considered, namely by the name of the relationships. In case a predicate is used to describe a relationship, a complete sentence may be the result. As demonstrated by the example, this is however not compulsory. The introduction of subject and predicate therefore depends on the discipline of the modeler. An ERM contains no control flow, so that it is not clear when and what actions are performed (predicate). Who the initiator of an action is, i.e., the subject, can only be concluded from the ER diagram when for the marking of relationships corresponding terms are used in a disciplined way.

14.3.2.3 Relational Data Model

For relational data models, analogous to the ERM, only data objects are considered, but here in the form of tables. Subject and predicate are accidentals.

As structural elements in relational data models, only those relations can be represented that can be described by tables. The rows of the tables are the data records, and the columns correspond to the data fields of the records. A data model usually consists of multiple tables. Relationships between any records, even in different tables in a model, can be constructed by using the same field content (primary and foreign keys).

Certain records are accessed via field contents. Figure 14.12 shows a data model for the business trip application. The data model consists of three tables "employees", "managers", and "business trip requests". The table "managers" includes all the supervisors; the table "employees" includes all employees with a reference to their managers in the column "M-No." The table "business trip



Managers

M-No.	Name	First Name
1	Schmid	Werner
2	Meier	Andreas
3	Miller	Anton
...	...	...

Employees

EM-No.	Name	First Name	M-No.
1	Schulz	Christoph	1
2	Dorfner	Hubert	1
3	Huber	Antonia	2
...	...	...	...

Bt-requests

Btr-No.	Start	End	EM-No.
1	12.3.2007	10.3.2007	1
2	12.7.2007	5.8.2007	1
3	4.9.2001	15.9.2007	3
...	...	...	

Fig. 14.12 Relational data model for the business trip application process

requests" includes all business trip requests submitted so far. The column "EM-No." in the table "business trip requests" contains a reference to the employee who has provided this business trip request.

On relational data models, logical, set-theoretic queries are defined (predicates) that are used by users (subjects). A relational data model does not include which users (subjects) are available in a certain situation or part of reality. The possible predicates that are triggered by the users are specified by the so-called query language, in general, the Structured Query Language (SQL).

In the example, the manager Werner Schmid (a user, subject) determines his subordinates by an appropriate query (predicate) from the "employees" table (objects). These are all the employees that contain a "1" in the column "M.-No." in the table "employees". Then, in the "business trip requests" table, all business trip requests are identified that contain in the "EM-No." column a number of an employee of Werner Schmid. The result set of this query therefore contains all the business trip requests of Mr. Schmid's employees, which can then be processed. Using the query language for relational databases, the predicate is present, while it is completely missing in the ERM.

Relational data models are very close to implementation. They can more or less be directly realized by a relational database, using ERM as a modeling language and the relational model already as a programming facility. In both modeling languages, however, subjects are only marginally considered. For a database application, there is always only "the" user, whoever that may be. The subject concept comes into play only in the context of authorization concepts: Which users can access which data in which way?

14.3.3 Modeling While Focusing on Predicate and Object

14.3.3.1 Origin

In the previously described modeling methods, either the subject or the predicate has been neglected. In the predicate-centered methods, the object aspect has been insufficiently described, in object-supporting methods, the predicate aspect. For databases, although there is a query language that can be used to form predicates, there is no way to define control flows (i.e., sequences of predicates). In the technical implementation of such incomplete models, missing components must be interpreted, which may lead to incorrect implementations.

It was natural, therefore, to develop modeling approaches considering action and data aspects in a balanced way, i.e., modeling languages, such as the data flow diagram, that contain predicates and objects. In this way, complete sentences can be formed in terms of the standard semantics of sentences, namely passive sentences. Passive sentences are used in natural languages, when the subject plays a minor role. A passive description of the business trip application process could be as follows: "The business trip application is filled out, the business trip request will be checked, the check result is documented, and the travel accounts of the employees (business trip directory) will be updated."

14.3.3.2 Data Flow Diagrams

Using data flow diagrams (DFD), the flow of data between functions, data repositories, and external stakeholders who are not part of the operation of the system are represented. The Structured Analysis by Tom DeMarco (DeMarco 1979) is an application of data flow diagrams for modeling.

In data flow diagrams, the following graphical elements are used:

- External interface (external partners, stakeholders, terminators): External interfaces are represented as rectangles. They denote the relations of the considered system to the outside world. They send or receive data, but do not process them. External interfaces trigger the system by the provision of data and can therefore be considered under certain restrictions as subjects.
- Function (process, task, function): Functions are shown as circles or ovals. They have the task of processing input into output data and contain the necessary algorithms. The functions correspond to predicates according to the semantics of natural language. Predicates of higher complexity can be refined by the predicates of a control flow diagram.
- Data storage (store, repository): Stores are presented as two parallel lines. They form a storage facility for data with different times of creation and use. They can be regarded as special data storage functions.
- Data flow (information flow, data flow): The data flow is represented by arrows between functions or data stores. The arrows are labeled with the name of the data flowing. In a data dictionary, the structures of all information items used are defined. The definition of data structures is done in Backus–Naur form. In this respect, an ERM could of course also be used. The data corresponds to the objects of the natural language sentence semantics.
- Context Diagram: Figure 14.13 shows the context diagram of the business trip application process. The context diagram identifies the external interfaces and illustrates the system to be developed as a function. The context diagram describes how the application receives data from an external interface and returns the result to the external interface. In this example, the external interface can be interpreted as a subject (employee). However, the manager is missing, since he is part of the system. If he and the update of the business trip data are also relocated (to the outside), virtually nothing remains from the application.

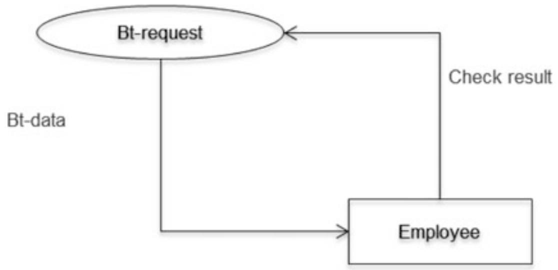


Fig. 14.13 Context diagram for the business trip application process

Figure 14.14 shows the refinement of the business trip process with the data flow between the individual functions and data stores. It is important to note that no control flow is connected to the data flow, although this might be suggested by the representation.

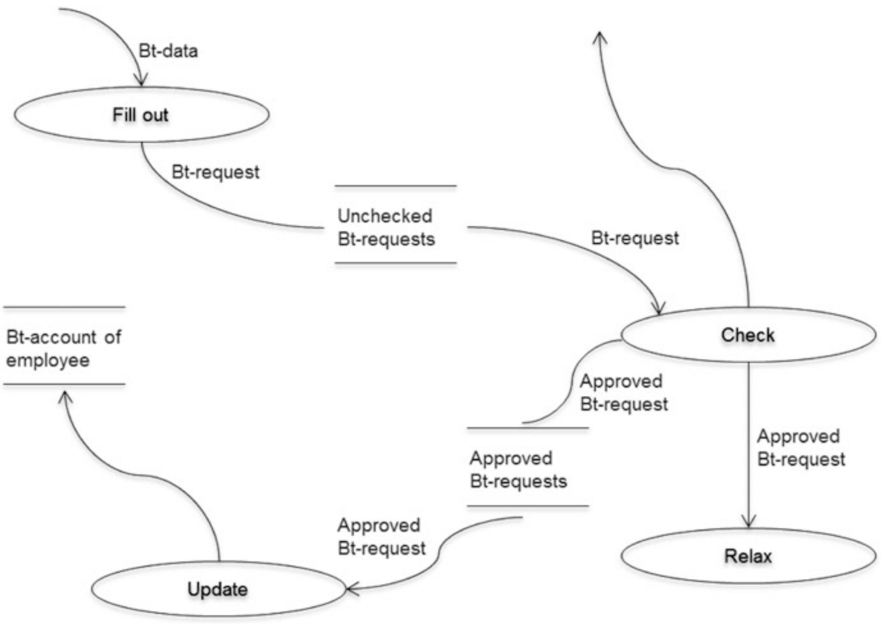


Fig. 14.14 Business trip application process as a data flow diagram

Although data flow diagrams were already developed in the 1970s, they cover predicate and object from the natural language sentence semantics. However, subjects can only be introduced via auxiliary constructions which lead to distortions. Data flow diagrams are no longer used in practice. The combination of predicate–object has evolved and led to object-oriented modeling and implementation methods.

14.3.3.3 Object Orientation

The basic idea of object-oriented programming is coupling functions (methods) that can be applied to data as closely as possible with the data being processed, including their properties, and to encapsulate them from the outside. The functions together with the data form an object in the sense of object-oriented modeling. The data of an object can only be accessed with its own methods. Objects with similar properties can be grouped into classes. Simple objects (or classes) can be developed by operations such as inheritance, polymorphism, aggregation, associations, etc. into complex structured objects and classes. For more details on the object-oriented methodology, we refer to the extensive existing literature (cf. http://www.uml.org).

Today, object orientation is the common standard for modeling and programming. Compared to approaches in which properties and functions are not considered in an integrated way, this modeling paradigm makes the claim of being able to represent the observable world more accurately than other approaches.

The object-oriented modeling approach, with objects consisting of data and functionality, covers the concepts of predicate and object according to the natural language sentence semantics. The functions correspond to the predicates and the data to the objects.

Figure 14.15 shows the object "business trip request" with the data "start of trip", "end of trip", and "check result" and the functions "fill out", "check", and "enter check result". In case the business trip is approved, the travel directory represented by the object "travel account" is updated.

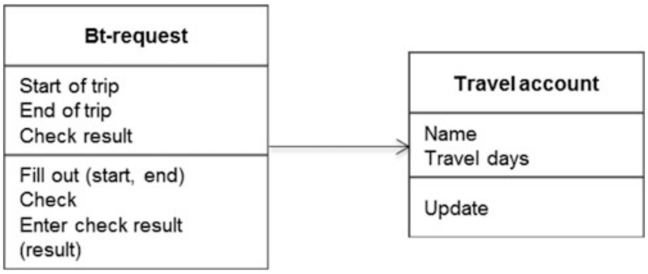


Fig. 14.15 Object or object class business trip request

The object "business trip request" now allows formulating incomplete sentences such as "fill out business trip request" or "check business trip request". To form complete sentences in the original object-oriented approaches, subjects could only be inserted into the model by natural language elements.

With the introduction of use case diagrams as contained in UML, this deficiency has been removed. UML has been developed by the Object Management Group (OMG) as a standardized language for modeling software and other systems. It includes 13 different types of diagrams (http://www.omg.org/spec/UML/2.2/). One of these diagram types is the use case diagram. The introduction of the subjects into the grammar of modeling by use case and activity diagrams will be discussed in Sect. 14.3.5.2.

14.3.4 Modeling While Focusing on Subjects

14.3.4.1 Origin

In computer science, there has long been the concept of parallel processes. A process executes actions within a given time interval to achieve a specific goal (Havey 2005). A process description defines the behavior of a process.

In the natural language sentence semantics, the subject is the starting point of activities defined by the predicate. Thus, subjects represent the active elements of reality. Subjects can execute defined sequences of actions (predicates). Subjects are mutually independent and communicate with each other, if required, i.e., they exchange information. Subjects, therefore, largely correspond to processes in computer science. Using the process concept, subjects from reality can be mapped to a corresponding construct in a model.

In the following sections, two concepts are introduced that put processes into the center of attention. For this purpose, parallel processes are defined which synchronize themselves through the exchange of messages, i.e., a process can send and receive messages by way of so-called ports. Sending and receiving are therefore the only possible predicates. Ports for message exchange can be interpreted as objects of the natural language sentence semantics.

14.3.4.2 Calculus of Communicating Systems

Calculus of Communicating Systems (CCS) is a process algebra (Milner 1980). A process algebra is used for algebraic modeling of parallel processes and consists of elementary actions and operators for joining actions. Elementary actions cannot be further detailed.

Processes can interact with the neighbors or independently perform activities in parallel. The aim of CCS is to model the communication between processes, e.g., to investigate their equivalence.

A process uses ports as enablers of communication with other processes, whereby each port has a name. A distinction is made between send and receive ports. Figure 14.16 shows the individual processes or subjects, respectively, of the business trip application process. The employee sends the business trip request to the manager. For the send port, the port name is marked with a horizontal line. The manager sends the result to the employee, and, where appropriate, the approved business trip request to the travel office.

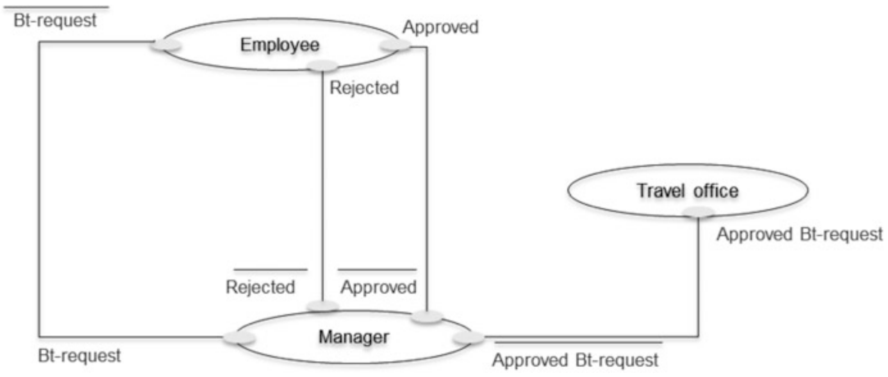

$\overline{\text{Bt-request}}$
Employee
Approved
Rejected
Travel office
Approved Bt-request
$\overline{\text{Rejected}}$
$\overline{\text{Approved}}$
Manager
Bt-request
$\overline{\text{Approved Bt-request}}$

Fig. 14.16 CCS processes for business trip request

In Figure 14.16, only the involved processes and their relationships are shown. The internal behavior is not yet visible. This is described using operators. In our example, we use only a few of these operators; for a complete list, we refer to the literature (Milner 1992; Milner et al. 1992a, b; Brinksma and Mader 2003). Figure 14.17 shows the behavioral description of the individual processes and their coupling to the business trip application process.

$$\begin{aligned}
&\text{Employee} = \overline{\text{Bt-request}} \,.\, (\text{rejected} + \text{approved}) \,.\, \text{NIL}\\
&\text{Manager} = \text{Bt-request} \,.\, (\overline{\text{rejected}} + \overline{\text{approved}} \,.\, \overline{\text{approved Bt-request}}) \,.\, \text{NIL}\\
&\text{Travel office} = \text{approved BT-request} \,.\, \text{NIL}
\end{aligned}$$

$$\textbf{Business trip application process = employee / manager / travel office}$$

Fig. 14.17 Description of the business trip application process in CCS

In the example, the process "employee" first sends the business trip request and then waits for either the message "rejected" or "approved". Once the employee receives one of these messages, the process can be continued. In case he performs the operation NIL, the process stops. The description of the processes "manager" and "travel office" can be interpreted similarly. The last line in the figure shows the composition of the entire process using the corresponding operator.

The business trip example shows that the active element in CCS, the actor, is seen as essential, while predicate and object play a subordinate role. Thus, CCS can be considered a subject-oriented method.

14.3.4.3 Communicating Sequential Processes

Communicating Sequential Processes (CSP) is also a process algebra. It was developed by Tony Hoare (1985). CSP was first published as a programming language construct and then formalized in the following years also due to the influence of Milner (1980). In CSP, in contrast to CCS, there is initially no distinction between sending and receiving. In case processes are linked by operators, also events of the same name from the associated processes are linked.

In Figure 14.18, the business trip application process is described in CSP. For employees, the event "business trip request" is enabled, and subsequently, either the event "rejected" or "approved". The event "SKIP" describes that the process is completed. In the process "manager", also the event "business trip request" is possible and then, appropriate follow-up events. When the process "employee" is linked to the process "manager" by using the ‖ operator (see last line), they share the initial event, and in both processes the corresponding transition (arrow in row 1 and 2) is executed.

$$\begin{aligned}
&\text{Employee} &&= \text{Bt-request} \Rightarrow (\text{rejected} \Rightarrow \text{SKIP} \mid \text{approved} \Rightarrow \text{SKIP})\\
&\text{Manager} &&= \text{Bt-request} \Rightarrow (\text{rejected} \Rightarrow \text{SKIP} \mid \text{approved} \Rightarrow \text{approval} \Rightarrow \text{SKIP})\\
&\text{Travel office} &&= \text{Approval} \Rightarrow \text{SKIP}
\end{aligned}$$

$$\textbf{Business trip application process = employee} \parallel \textbf{manager} \parallel \textbf{travel office}$$

Fig. 14.18 Description of the business trip application process in CSP

On a detailed level of CSP, it is possible to dissolve events into send and receive operations that run on ports and can transfer data. In this way, in CSP, the predicates “send” and “receive” exist, as well as objects (messages) on which these (simple) predicates can be executed.

In CSP, analogously to CCS, the subject represents the essential part. Predicate and object play a very subordinate role. Without natural language additions with respect to predicate and object, a complete model of the business trip application process cannot be created with CSP. Meaningful names are also essential for understanding processes but do not contribute to the semantics.

14.3.5 Methods Considering Subject, Predicate, and Object

14.3.5.1 Origin

In all major formal modeling methods of computer science, natural language sentences cannot be formed in the sense of natural language. Since this is always necessary for achieving a thorough understanding, the missing elements have been informally added. For instance, the rectangles for the actions in flowcharts were labeled accordingly. Instead of “fill out”, the phrase “fill out business trip request” was used for labeling the action symbol. In English literature, such constructs are termed “verb–noun phrase” (Sharp and McDermott 2009, p. 45).

14.3.5.2 Use Case and Activity Diagrams in UML

UML has 13 diagram types. These are divided into six structural diagram types and seven behavior diagram types. Using the behavior diagrams, dynamic aspects of a program are described. The structure diagram types overlap in their representation aspects, whereby mutual systematic transfer is not possible. All seven diagram types include aspects of subjects, however, in an unclear form. In UML, all entities of discourse are objects. In the following, those diagram types in which the subject aspect most clearly comes to light are explained in more detail. These are the Use Case Diagram and the Activity Diagram.

Use Case Diagrams allow describing the use of a system from a user perspective. A use case shows which users (actors = subject) perform what actions (predicates) using the system. A use case describes the externally visible behavior of the considered element (system, class, etc.) and encapsulates a coherent set of actions that are executed in a fixed order. A use case does not indicate which classes and which individual operations on the actions are involved. A description of the use case is complete once the underlying processes are defined. To accomplish this, an appropriate method of UML for modeling behavior, or a natural language description, can be used.

Actors are considered special UML classes with specific properties and are not considered as being definitely active. It can therefore only be determined which actions occur between an actor and the system, but not who is the starting point of an action. However, it is advisable to consider an actor as the starting point of actions.

Figure 14.19 shows the Use Case Diagram for the business trip application process. The complete sequence of actions for "fill out request" could mean: "enter start date of business trip", "add business trip end date", and "ask manager for decision". The other use cases can be described analogously.

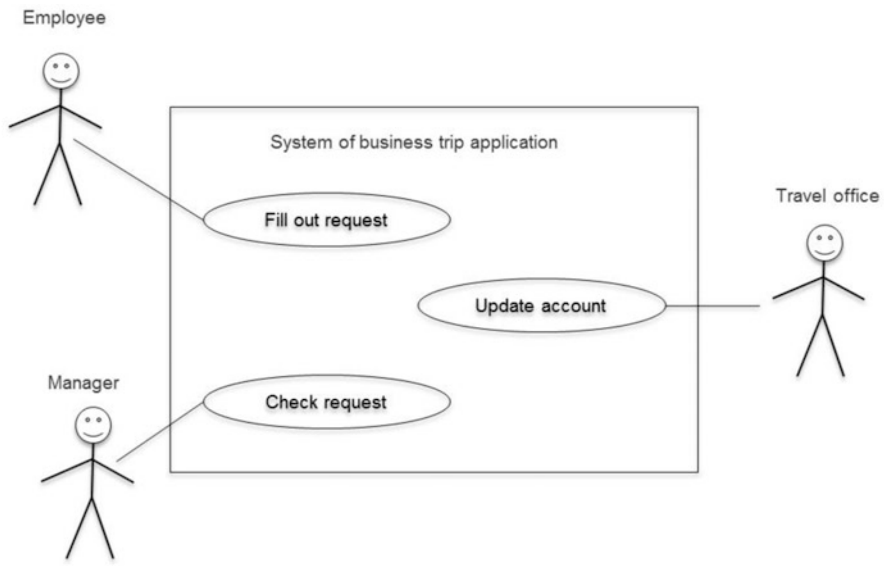


Fig. 14.19 Use case diagram for the business trip application process

Use Case Diagrams are often refined further by using activity diagrams in which elements of data flow diagrams, Petri nets, flowcharts, etc. are combined. However, the interplay of several activity diagrams by means of modeling signals and events for exchanging information is only rudimentarily possible. This means that representing the relationship between the individual use cases in our example is not possible at all on the level of Use Case Diagrams and only to a limited extent on the level of activity diagrams. An example in this respect is the alternative waiting of an employee for approval or rejection.

The following example shows an activity diagram for the business trip application (see Fig. 14.20). The individual activities have been grouped with so-called swim lanes, depending on who performs the activity. In our example, there is a dedicated swim lane for the employee, the manager, and the travel office. These lanes can be considered as subjects who carry out the assigned activities. The sequence of activities is specified by the control flow analogously to flowcharts.

It is possible to split up a single control flow by fork and join operations into parallel control flows (fork) and to rejoin them again (join). In the business trip application example, the control flow is split after the approval of the request by the manager (shown in the picture with a black bar in the swim lane of the manager). This means that the employee and the travel office obtain the approval in parallel. The parallel control flows are then joined before the end node is reached.

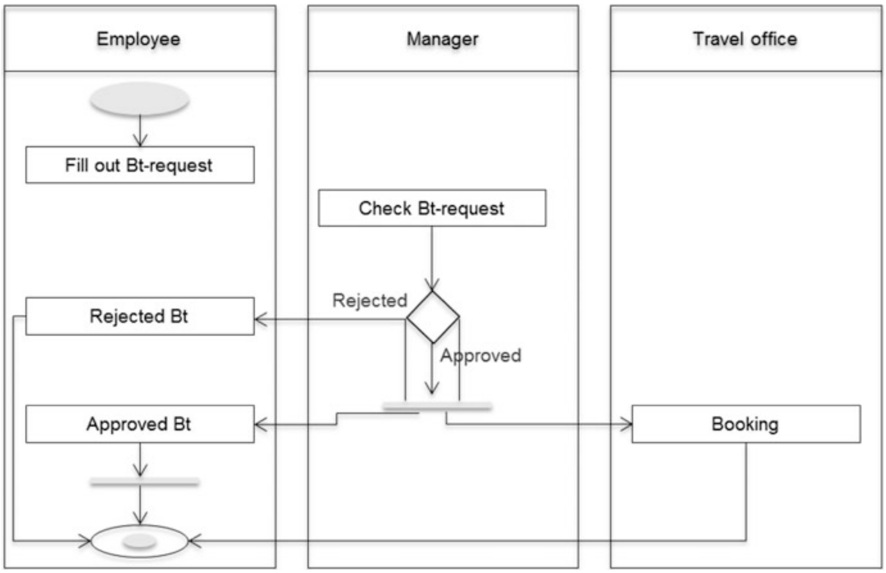


Fig. 14.20 Activity diagram of the business trip application process

The coordination of individual activities is done by shifting the control flow between the individual lanes. However, it seems unrealistic that the control flow, after completion of the business trip request by the employee, changes without further delay to the manager. Normally, process participants exchange messages when transferring the control flow. Such a transition of the control flow from one process participant to another is not obvious, and visible only with cognitive effort in an Activity Diagram.

In addition, fork and join operations in a neighboring swim lane are elusive and artificial. In fact, they are often omitted, which is even officially allowed in BPMN (http://www.omg.org/spec/BPMN/2.0) but immediately leads to semantic difficulties when using Fork, and especially Join.

Despite the identified shortcomings, UML provides with use case and activity diagrams and the other diagram types at least a limited possibility of complete sentence construction in terms of the standard sentence grammar. In UML, actors are not part of the model, so their behavior, and in particular the potential communication among stakeholders, is not considered in detail. This is also evident from the fact that the actors do not appear in the other diagram types in UML, with the exception of the time-sequence diagram.

Since the actors play an important role in business processes, UML also represents in models only a limited perspective on reality.

14.3.5.3 A Subject-Oriented Approach Using PASS

The subject-oriented methodology presented mainly in Chap. 5 of this book is based on the Parallel Activity Specification Scheme (PASS) of Fleischmann (1994).

PASS uses elements of the Calculus of Communicating Systems by Milner and the Communicating Sequential Processes by Hoare (see Sects. 14.3.4.2 and 14.3.4.3). It integrates aspects of object orientation and adds a graphical notation (cf. Schmidt et al. 2009, p. 54). In this way, S-BPM takes into account all parts of the natural language sentence semantics, including subject, predicate, and object, whereas the subject is in the role of "primus inter pares".

14.3.6 Synopsis

The table in Fig. 14.21 summarizes the findings from the previous sections. The more or less filled circle symbols express the assessment of various methods in terms of their coverage of the standard sentence semantics of natural languages. The table shows that parts of semantics are absent in many methods. We have demonstrated that these are added pragmatically by natural language comments, or by extending the basic set of symbols, to be able to form complete sentences.

Subject-oriented modeling targets active subjects (actors) and assigns activities and business objects either to them, or to their communication relationships. It thus meets the requirements of standard sentence semantics of natural language in its originally conceived sequence. Therefore, it is the only approach which can be considered complete in this respect. In addition, subject-oriented modeling is intuitive: it reduces the learning curve for modeling to the effort required for acquiring and mastering sentences of natural language.

Modeling language	**Looking at**			**Explanation**
	Subject	Predicate	Object	
Natural language	●	●	●	Description of the facts by whole sentences (who does what to what?), Exception: passive.
Control flow diagram	○	●	○	Focus on algorithms; usually natural-language extensions for objects and subjects.
(extended) Event-driven process chain	⊘	●	⊘	For EPCs, only predicates; extended EPCs allow the addition of subjects and objects.
Petri nets	○	●	⊘	Only predicates; subjects and objects by natural-language extensions.
Entity Relationship Diagram	○	○	●	Only the structure of the object is described. No operations are possible.
Relational databases	○	⊘	●	Tables as objects and SQL as language for predicates.
Data flow diagam	○	●	●	Data and data flows between various action points.
Object-oriented approaches	○	●	●	Predicates correspond to the methods that are defined on the respective data.
Calculus of Communicating Systems (CCS)	●	⊘	○	Process algebra for modeling parallel systems. As a single operation, there are "send" and "receive".
Communicating Sequential Processes (CSP)	●	⊘	○	Formal method for describing parallel systems that synchronize by the exchange of messages.
Unified Modeling Language	⊘	●	●	Originally intended only for describing object-oriented systems (predicates and objects). In the use case diagrams, there are stakeholders as actors / subjects and the subjects of activity diagrams can be defined by swim lanes.
S-BPM based on PASS	●	●	●	Subjects interact with each other. They exchange messages and transmit thus business objects, which they edit internally.

Fig. 14.21 Model description languages in comparison with respect to standard semantics structure of sentences (based on Schmidt et al. 2009, p. 55)

References

Brinksma, E., Mader, A. Prozessalgebra, Teil 1, in: at - Automatisierungstechnik, Vol. 51, Issue 8, S. A13–A16, 2003.

DeMarco, T., Structured Analysis and System Specification, Upper Saddle River 1979.

Denert, E., Software-Engineering - Methodische Projektabwicklung, Berlin 1991.

Fleischmann, A., Distributed Systems – Software Design and Implementation, Berlin 1994.

Havey, M., Essential Business Process Modeling, Sebastopol 2005.

Hoare, C., Communicating Sequential Processes, New Jersey 1985.

Milner, R., Calculus of Communicating Systems, Berlin u.a. 1980.

Milner, R., Parrow, J., Walker, D., A Calculus for Mobile Processes, Part I. Information and Computation 100, pp. 1–40, 1992.

Milner, R., Parrow, J., Walker, D., A Calculus for Mobile Processes, Part II. Information and Computation 100, pp. 41–77, 1992.

Milner, R., Functions as processes, in: Math. Struct. in Comp. Science, vol. 2, pp. 119–141, 1992.

Sharp, A., McDermott, P., Workflow Modeling, Norwood 2009.

Scholz, M., Holl, A., Objektorientierung und Poppers Drei-Welten-Modell als Theoriekerne, in: Schütte, R. et al., Wirtschaftsinformatik und Wissenschaftstheorie. Grundpositionen und Theoriekerne, Arbeitsbericht 4 des Instituts für Produktion und industrielles Informationsmanagement an der Universität Essen, Essen 1999.

Scheer A.-W., ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, Berlin 1998.

Schmidt, W., Fleischmann, A. und Gilbert, O., Subjektorientiertes Geschäftsprozessmanagement, HMD – Praxis der Wirtschaftsinformatik, Heft 266, S. 52–62, 2009.

Sinz, E.J., Konstruktionsforschung in der Wirtschaftsinformatik: Was sind die Erkenntnisziele gestaltungsorientierter Wirtschaftsinformatik-Forschung?, in: Österle, H., Winter. R., Brenner, W. (Hrsg.), Gestaltungsorientierte Wirtschaftsinformatik: Ein Plädoyer für Rigor und Relevanz, Nürnberg, S. 29–34, 2010.

Stucky, W., Winand, U. (Hrsg.), Petri-Netze zur Modellierung verteilter DV-Systeme – Erfahrungen im Rahmen des DFG-Schwerpunktprogramms "Verteilte DV-Systeme in der Betriebswirtschaft", Bericht 350, Karlsruhe 1997.