2. Background of Software and Business Process Modeling

This chapter introduces the term of modeling especially in the domains of software engineering and business process management. Therefore, the general concept of modeling and the history of modeling in enterprises is subsumed and introduced. Furthermore, important and for this thesis relevant terms in the modeling domain are defined and pictured regarding their semantic correspondence. The survey in this thesis aims to be a cross-language survey over different modeling concepts. Therefore, the similarities of different modeling concepts and consequently the similarity of resulting languages are defined. In a nutshell, the following chapter shows the general view and understanding of modeling applied in this thesis.

2.1. General Concept of Modeling

The term “model” is derived from the Latin word modulus, which means measure, rule or pattern. Obvious examples are toy railways and dolls, maps as well as architectural models of buildings. In the domain of enterprise modeling, process models, design patterns and architectural diagrams exist. Other models are less obvious such as project plans, specifications, designs and metrics [Ludewig, 2003].

In order to distinguish models from other artefacts, specific criteria are needed. According to Stachowiak’s Model Theory, any candidate must fulfil three criteria being a model [Stachowiak, 1973]:

- **Mapping criterion**: there is an original object or phenomenon that is mapped to the model. This original object or phenomenon is referred to as “the original”.

- **Reduction criterion**: not all the properties of the original are mapped on to the model, but the model is somehow reduced. On the other hand, the model must mirror at least some properties of the original.

- **Pragmatic criterion**: the model can replace the original for some purpose, i.e. the model is useful.
10

2 Background of Software and Business Process Modeling

The **mapping criterion** does not imply the actual existence of the original; it may be planned, suspected, or fictitious. The cost estimation of a software project is a speculative model of the future. A model may act as the original of another model. A program design is a model of the code to be written, while the code is a model of the computation performed by the computer when the code is executed.

At first glance, the **reduction criterion** seems to describe a weakness of models, because something is lost in the model that was present in the original. But that loss is the real strength of models: very often, the model can be handled while the original cannot.

Additionally, the **pragmatic criterion** is the reason models are applied. Since it is not possible to use the original, the model is used instead.

As an effect of the reduction, many features of the original (the waived attributes) are not found in the model. For example, the name of a person is not visible in his photograph. On the other hand, features that do not stem from the original are added (abundant attributes). For example, the size of the picture does not tell anything about the person.

Figure 2.1.1 shows the resulting relation between a original real-world domain and its associated model.

However, a weakness of Stachowiak’s concept of a model is that it implies an epistemological position of positivism. This is criticized in Schuette & Rotthowe (1998), where the authors propose an alternative position based on insights from critical realism and constructivism. This position regards a model as a “result of a construct done by a modeler” [Schuette and Rotthowe, 1998].

Consequently, a relationship between a model and the modeler exists leading to limitations of resulting models, which depend on the modeler’s subjective view of modeling and the domain modeled [Krogstie, 2003].

As such, it is heavily influenced by the subjective perception of the modeler. This makes modeling a non-deterministic task that requires standards in order to achieve a certain level of
inter-subjectivity.

A **graphical modeling language** meets this demand for standardization in the domain of software and business process modeling.

### 2.2. History of Conceptual Modeling

Conceptual modeling plays an important role in the development of software applications. Modeling acts as a starting point for understanding the common basis for developers and users. Conceptual modeling integrates domain experts, who are involved in a business process, and their knowledge, into the software development.

In the following chapter a brief history of conceptual modeling is proposed starting with the roots in the early 70ies and ending with the object oriented development and related modeling languages.

#### The Beginning of conceptual modeling in the early 1970ies

One of the first approaches to abstraction applied to software engineering can be found in [Parnas, 1972].

The major goal of this approach is to provide precise and complete application specifications leading to the fact that other pieces of software can be developed to interact with the application without additional information. This is achieved by providing concepts for data abstraction and hiding implementation details from the user.

Codd (1970) separated logical data organization from physical organization. This approach laid the ground for conceptual modeling and for capturing the semantics of an application [Codd, 1970].

Another important approach in the field of modeling was the programming language Simula. Simula introduced new concepts like objects, classes, methods and especially subclasses, which support the notion of generalization abstraction. Simula is considered the first object-oriented programming language and became a cornerstone of most object-oriented techniques [Dahl and Nygaard, 1966].

#### Conceptual modeling and Semantic Data Models in the mid 1970s

Most approaches to database design relied on modeling data structures, which are used to store the model in file systems. Two approaches are the hierarchical and the network models,
both focusing on the physical level what nowadays may be called graph models [Angles and Guteierrez, 2008].

Techniques for knowledge integration as well as data abstraction were introduced by Abrial (1974). He proposed a definition of the semantics of classes by access procedures [Abrial, 1974].

Because of its simplicity, Chen’s Entity-Relationship Diagram became popular and the de facto standard in data modeling and database design [Chen, 1976].

Semantic data models allow for designing models at a higher level and enable the database practitioners to naturally and directly incorporate in the schema a larger portion of the semantics of the data [Hammer and McLeod, 1978].

The Structured Analysis and Design Technique (SADT), introduced by Ross (1977) in the mid seventies, was one of the most significant early steps in the area of requirements specifications [Ross, 1977]. Among the features is the emphasis on modeling data as well as activities connected by edges representing the flow of information.

Smith and Smith (1977) have introduced concepts for abstraction and generalization in database design in database research. They combined ‘generalization’ and ‘aggregation’ into one structuring discipline. However, aggregation was still not easily modeled using ER; it became the main thrust in Object Oriented databases. The advantage of aggregation is that it provides an easier understanding of complex models and a more systematic approach to database design. It mainly supports the development of highly structured models without loss in intellectual manageability [Smith and Smith, 1977].

The first high level data definition languages for defining conceptual schemas such as the Conceptual Schema Language were discussed in the late seventies. Descriptive elements as well as procedural elements are provided within this language.

Hence, static aspects and dynamic behavior of data could be described by providing standard types, object types and association types. A prominent example of a database design language covering the concepts is Taxis [Mylopoulos et al., 1978]. Taxis provides relational database management facilities, means of specifying semantic integrity constraints incorporated into transactions, and an exception-handling mechanism. Taxis applies the concepts of class, property and generalization relationship to all aspects of program design.

**An Efflorescence of conceptual modeling languages in the 1980s**

The specialization has increased over the years and more and more sub-disciplines within conceptual modeling emerged. The differences between those sub-disciplines seem to arise from issues concerning notation and basic vocabulary. Only in a minority of cases, the ways to utilise the models or the ways the models are constructed justify this development.
In the domain of requirements engineering, Greenspan et al. (1994) adopted the approach of Taxis and attempted to formalize the SADT notation [Greenspan et al., 1994]. The Requirements Modeling Language embodies a notation for requirements modeling which combines object-orientation and organization, with an assertional sublanguage used to specify constraints and deductive rules [Roussopoulos and Yeh, 1984].

The Semantic Database Design takes not only the environment of the data processing system into account, but also focuses on the environment of the entire company. This approach proceeds on the assumption that without a complete understanding of how the enterprise operates, it is not possible to develop an effective design. Thus, it is recommended to start with an environment analysis phase followed by a system analysis phase, capturing and analyzing the operational behavior of an organisation.

**Object-oriented development in the 1990s**

In the early 1990s the term process became a new productivity paradigm. Companies were encouraged to think in processes instead of functions and procedures. The Event Driven Process Chain (EPC) notation offers many ways for modeling processes, analyzing them, and identifying improvement potentials. EPCs were invented back in 1992 by Prof. Scheer and colleagues at the University of Saarland. Since then, they have seen an industry wide adoption. EPCs are used in many industries and are supported by different modeling tools.

Furthermore, a variety of object-oriented analysis techniques has been developed in the early nineties. Important representatives of these techniques are the ‘Booch Method’ and Rumbaugh’s ‘Object modeling Technique’, both offering a more coherent modeling framework than the combined use of data flow and Entity-Relationship diagrams. The Booch Method focused mainly on object-oriented design, whereas the object-modeling technique focused on object-oriented analysis.

In 1994 Booch and Rumbaugh decided to combine and unify their object-oriented modeling methods by developing the Unified modeling Language (UML) - a language for modeling object systems [OMG, 2011c].

Through the standardization efforts undertaken by the Object Management Group (OMG), UML has been rapidly adopted as a standard for modeling a very wide range of applications and domains [Booch et al., 1998].

It is claimed that one important advantage of UML is that it could be used both for modeling software and for modeling the problem domain that is supported by a system [Evermann and Wand, 2005].

By the end of the nineties it was widely agreed that information systems need to better match their operational organizational environment. Hence, requirement specification needs to cover not only software specifications but also business models and other kinds of information in
describing the context in which the intended system will function. The above UML emphasizes concepts for modeling and analysis during the later requirements phases, which usually focus on completeness, consistency, and automated verification of functional requirements.

With Tropos, a development method supporting the early phases of requirement engineering is provided. Tropos is founded on the idea of using the agent paradigm and related mentalistic notions during all phases of the development software process [Bresciani et al., 2004].

2.3. Terms and Definitions in the Modeling Domain

This chapter focuses on the introduction of terms and definitions in the modeling domain. The generic term in the domain of graphical modeling languages is modeling method. Modeling methods provide the necessary concepts capable to capture relevant domain knowledge in terms of models that describe relevant aspects of the application domain. Modeling methods consist of two basic components [Atkinson and Kuhne, 2003]:

- a *modeling technique*, which is divided in a modeling language and a modeling procedure
- *mechanisms* and *algorithms* working on the models described by the modeling language

The *modeling language* contains the elements, with which a model can be described. The *modeling procedure* describes the steps applying the modeling language to develop models.

A graphical modeling language is described by *syntax*, *semantics* and *a notation*. The syntax describes the elements and rules for creating models and is based on a grammar. The semantics describes the meaning of a modeling language and consists of a semantic domain and a semantic mapping. The semantic domain describes the meaning by using ontologies and mathematical expressions etc. The semantic mapping connects the syntactical constructs with their meaning defined in the semantic domain, i.e. semantic schema. For the formalization of semantic definitions, denotational semantics, operational semantics, axiomatic semantics or algebraic semantics are used [Petre, 2006].

The *notation* describes the visualization of the modeling language. Static approaches define the geometric shapes for visualizing the syntactical constructs, but they do not consider the state of the modeling constructs during modeling. Dynamic approaches consider the model state by splitting the notation in a representation part and a control part. The representation part maps to the static approach. The control part defines rules to query the model state and to influence the representation depending on the particular model state [Karagiannis and Kühn, 2002].

Generally, a model is developed by using a *modeling tool*. The tool represents the modeling language and may represent a modeling technique or a modeling method. A model describes a system or a system component such as application systems or business processes.
2.4. Abstraction Layers

To capture the complexity in software and business process modeling, different abstraction concepts are introduced. A traditional abstraction concept in computer science is the separation of modeling levels, from instance level to model level to metamodel level, denoted by horizontal abstraction.

For example, along the lines of the levels of abstraction identified by the OMG,

- the metamodel level,
- the model level,
- and the instance level

Figure 2.3.1.: Terminology of the modeling domain
play important roles in the design and analysis of complex systems in general and software systems in particular.

It is instructive to explain these levels in a bottom-up order, starting with the instance level. The instance level reflects the concrete entities that are involved in business processes or applications. Executed activities, concrete data values, and resources and persons are represented at the instance level. To organize the complexity of the real-world scenarios, a set of similar entities at the instance level are identified and classified at the model level.

For instance, a set of similar business process instances are classified and represented by a business process model. In object modeling, a set of similar entities is represented by a class, and in data modeling using the Entity Relationship approach, a set of similar entities is represented by an entity type, and similar relationships between entity types are represented by a relationship type.

2.5. Modeling in Software Engineering

In the field of Software Engineering modeling became popular with Peter Chen’s Entity-Relationship Diagram [Chen, 1976].

Since object orientation has evolved in the early 1990ies, modeling has become a self-contained and growing area in the domain of Software Engineering. Following the wide dissemination of the UML, the Model Driven Development and Model Driven Architecture approach has moved modeling into the centre of the software development process.
In this thesis, the UML in the version 2.0 plays a significant role due to its current ubiquity in the domain of software engineering. The UML is a standardized modeling language in the field of object-oriented software engineering. The standard was created and is managed by the OMG.

UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. UML diagrams represent two different types of a system model [OMG, 2011c]:

- **Static or structural models**: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.

- **Dynamic or behavioral models**: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

The following figure shows an overall view of the diagrams of the UML:

Models can be found in all areas and applications of software engineering. While software developers create concrete models people who do research in software engineering work on notations and methods for developing such concrete models. Class Diagrams and State Charts, Petri Nets and Data Flow Diagrams are a few examples of models that use such notations. Most of the models used in software engineering are prescriptive, for instance:

- process models, such as UML Activity Diagrams
- information flow models such as the diagrams used in structured Analysis and Design
2 Background of Software and Business Process Modeling

Figure 2.5.2.: Elements of UML Activity Diagrams [OMG, 2011d]

In the following sections significant modeling languages of the software engineering domain that are relevant in the forthcoming empirical surveys of this thesis are introduced.

2.5.1. UML 2.0 Activity Diagrams

The UML 2.0 Activity Diagram belongs to the behavioral diagrams of the UML and was designed for modeling processes and flows in application systems [OMG, 2011c].

The diagram is approximated on Petri Nets, and uses also the notion of token. Main concepts of the Activity Diagram are actions and activity partitions.

An activity partition is used to group actions executed by a certain role.

The main elements of a UML Activity Diagram is the activity and different activity nodes. Action, object node and control node are a specialization of an activity node.

Action describes the atomic task of an Activity Diagram. An object node is an activity node that indicates an instance of a particular classifier, possibly in a particular state, may be available at a particular point in the activity [OMG, 2011d].

Control nodes define the behavior of an Activity Diagram.

The initial node starts the activity. If an activity contains more initial nodes, different flows are executed concurrently.

The final node is split up into activity final node and flow final node. While the activity final node terminates all flows within an activity, the flow final node only terminates one flow, and the activity is unaffected.

The fork node splits the flow into concurrent paths. A fork node has one incoming flow and two ore more outgoing flows. A join node has two or more incoming flows and one outgoing flow.
Usability Evaluation of Modeling Languages
Schalles, C.
2013, XXIII, 183 p., Softcover
ISBN: 978-3-658-00050-9