Frank, Ulrich

Research Report

Multi-perspective enterprise modelling: Background and terminological foundation

ICB-Research Report, No. 46

Provided in Cooperation with:
University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB)

Suggested Citation: Frank, Ulrich (2011) : Multi-perspective enterprise modelling: Background and terminological foundation, ICB-Research Report, No. 46, Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB), Essen

This Version is available at:
https://hdl.handle.net/10419/70904

ICB
Institut für Informatik und Wirtschaftsinformatik

Ulrich Frank

Multi-Perspective Enterprise Modelling: Background and Terminological Foundation

ICB-RESEARCH REPORT

UNIVERSITÄT DUISBURG ESSEN

ICB-Research Report No. 46
December 2011

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Authors' Address:

Ulrich Frank

Lehrstuhl für Wirtschaftsinformatik
und Unternehmensmodellierung

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

ulrich.frank@uni-due.de

Contact:

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen

Tel.: 0201-183-4041
Fax: 0201-183-4011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)

"A surprisingly large number of terms in the information system area have been coined by suppliers of IT products and services, or by pseudo scientists."

The FRISCO Report

Abstract

In Information Systems, enterprise modelling has been a pivotal field of research that has evolved over a period of more than 20 years. In recent years, the main research focus was on adapting approaches to enterprise modelling to changing requirements, e.g. provide support for IT management. To develop a better understanding of the motivation, objectives and concepts that are characteristic for enterprise modelling, it is important to study the terminological background. On the one hand, it builds the foundation for the conceptualisation of enterprise models, on the other hand, it helps to clarify the semantic relation of the term "enterprise model" to terms in its surroundings such as "information system", "conceptual model" etc. This report is aimed at developing conceptualisations of respective terms. Starting with more generic terms like "domain", "model" or "information system", the course of the investigation gradually moves forward to "conceptual model", "action system", and "enterprise model". Finally, an elaborate conception of multi-perspective enterprise models is presented that is intended to characteristic aspects of their structure, the intentions related to them and preconditions of their reflective use.

Table of Contents

Figures

Tables

1 Introduction

Language is the essential tool of all scientific investigation. It provides us with concepts that allow us to structure a subject of interest, i. e. to conceptualize and analyze it. Like any other tool it will often be adapted to the task – and we need to adopt it. In Information Systems ("Wirtschaftsinformatik"), language is of outstanding importance for a further reason. Information systems are linguistic artefacts. Hence, designing and using them requires appropriate languages. The design and use of information systems will often involve people that speak different technical languages, which results in the need to foster communication between these different parties – by providing appropriate languages. In the area of research that forms the background of this report – enterprise modelling – language, especially domain-specific modelling languages (DSML), is the core research topic. This includes both, the thorough analysis of existing technical languages with respect to certain purposes and the (re-) construction of new languages. For such a field of research, it seems obligatory to be especially demanding when it comes to its own terminological foundation. However, core terms such as "model", "enterprise model", "integration", "method" etc. are often not precisely defined. This may be contributed to the impression that the meaning of these basic terms is widely evident. Unfortunately, this impression is inappropriate. This report was originally intended to provide my students with a glossary of key terms in the area of enterprise modelling. However, my early attempts to develop brief and concise definitions of core concepts lead to the insight that this would – in many cases – require simplifications that might compromise a deeper understanding. Therefore I decided to aim at more elaborate conceptions – at the risk to alienate those who would rather expect a glossary of terms that could serve as an authoritative reference.

2 Domains, Models – and Abstraction

The term DSML implies the description of a corresponding *domain*. Examples of domains include application areas of certain classes of software systems, e.g. data warehouse systems or geographical information systems. Other domains are characterized by technical systems, e.g. automobiles or aeroplanes, that are in part controlled by software. Further domains may be constituted by the peculiarities of certain software systems, e.g. the domain of developing video games. As with other basic terms that we use every day, reflecting upon the term "domain" produces an ambivalent insight: It seems that there is no need for a clarifying definition because everybody seems to have an appropriate understanding of the term. At the same time, a comprehensive definition seems hardly possible, because it would require referring to other concepts, the meaning of which is more precise – and more common. Against this background the following description is rather an attempt to characterize the term than to define it.

> A **domain** is a subject area or a field of interest. It is not restricted to existing objects or phenomena, but may also include potential objects or phenomena. A domain can be characterized by the objects it includes, features of these objects, or functions provided by the objects. Objects include physical objects, human actors, and immaterial objects.

With respect to our key topic – modelling – it is important to realize that thinking of a domain will usually imply abstraction: To characterize it, we need to identify features we are interested in and/or fade out those that are of no relevance. Regarding a domain as an abstraction has two important implications: First, it means that a domain may cover a range of actual and potential systems that are characterized by common features. Second, it means that the level of abstraction related to the conception of a domain may vary substantially. Note that our conception of domain is clearly different from the use of the term in database theory, where it represents a set of uniform data that serves the specification of attributes.

Even more than "domain", "model" is a pivotal term in our field – and in many other fields, too. Although many textbooks provide generic definitions, a closer look reveals that a comprehensive definition is hardly possible. One reason for that is the heterogeneous use of the term in different disciplines. According to a prevalent definition often cited in various disciplines (see exemplarily (Stachowiak 1973)) a model is an *abstraction* that was created with a certain *purpose* in mind. Purposes include analysis, planning, forecasting, explanation, design, decision making etc. Often, such a conception is related to the idea of an original that is *represented* by the model. The act of abstraction is often seen as a deliberate simplification by fading out certain properties of the original while others are mapped to the model. Such a conception of "model" works fine in many cases. However, it is not entirely convincing. This is for various interrelated reasons:

No clear distinction of model and original: The prevalent conception of model is built on the assumption that a model can be clearly distinguished from an original – and it suggests that a model can be identified as such. However, this is not necessarily the case. Mahr points to the fact that any (physical) object can be regarded as a model, in the sense of a prototypical instantiation. At the same time, no "thing" has to be a model, because being a model is not an inherent feature, but depends on an interpretation – and there may be interpretations that do not regard a "thing" as a model that may from other perspectives look as such ((Mahr 2009), p. 232).

Naïve realism: It seems perfectly in line with the common understanding of modelling to regard a model as the result of representing selected aspects of an original. Nevertheless, such a conception reflects a naïve view of the problem. It includes the assumption that the considered domain has an objective existence of its own independent of its observer (ontological presupposition) and the assumption that we are able to perceive it as it actually is (epistemological presupposition). This position, referred to as "naïve realism", does not account for three aspects. First, a domain can hardly be regarded as an original in the sense of something real. Instead, the conception of a domain itself is based on an abstraction (see above). Second,

the domains we consider include patterns of (social) action (see below). Even if one assumes that they exist independently from the observer (which is, of course, impossible to prove), it is hard to imagine that one can perceive and conceptualize them as they actually are. While this argument may appear as a mere philosophical sophistry, it has a clear impact on the way we develop, interpret and evaluate models. This will become clearer when we account for action systems and the role of language (see below). It remains to add that this critique does not apply to Stachowiak even though he is often used as a reference for justifying a naïve conception of models. Instead, he is an explicit opponent of naïve realism and regards "reality" as a construction ((Stachowiak 1973), p. 285, (Stachowiak 1983)).

Mapping as a problem: The idea of mapping selected features of an original to a model reveals a further problematic implication of naïve realism. First, it is usually accompanied by a constraint that seems reasonable: Not any mapping should result in a model. Instead, it should preserve characteristic features of the original. In other words: It should resemble the original. Usually, this quality constraint is related to the idea of a "homomorphous" mapping which results in a model that preserves the structure of the original. However, as long as we cannot prove that our perception of the original is correct and complete, it is impossible to prove that a mapping is homomorphous. In a strict sense, this implies that any mapping can be claimed as homomorphous without the risk of refutation ((Zelewski 1995), p. 25).

Lack of an "original": Often, there is no original in a strict sense. Instead, we focus on domains which depend on abstractions (see above). In addition to that, we may sometimes want to overcome the limitations of existing practice. As a consequence, a corresponding model does not represent something existing on purpose. As we shall see, this aspect is of particular relevance for modelling in the realm of Information Systems.

Mainly as a reaction to the problem of distinguishing a model per se from an original, Mahr concludes that traditional techniques of disambiguation ("Begriffsklärung") do not allow for answering the question "What is a model?" ((Mahr 2009), p. 232). Instead, he suggests that the interpretation of a "thing" as a model should depend on an informed judgment about "model being" ("Modellsein", ibid, p. 235). Other authors aim at a concept of model that avoids the fallacies caused by naïve realism. They emphasize that the idea of mapping from an original is not appropriate. Instead, they suggest regarding models as *constructions* (e.g. (Klein and Lyytinen 1992), (Floyd 1992). Ortner speaks of constructions, too, but uses the term "artefact" rather than "model" ((Ortner 1997), p. 11). Schütte emphasizes an important feature of constructions to qualify as a model – in correspondence to Mahr's later suggestion: Following up on a line of thought presented by ((Stegmüller 1986), p. 15), he speaks of a model as a construction that is explicitly declared as a model ((Schütte 2000), p. 6).

Against this background, I suggest a preliminary definition of features that are characteristic for an artefact to serve as a model. It is not restricted to the assumption that there is an original. Instead, it is assumed that a model is related to a domain. As a consequence, it does not reduce a model to a mapping – even in cases where an "original" exists.

> A **model** in general is a *construction* that results from purposeful abstractions of a domain.

It may seem that mental models are not covered by this definition, since they seem not to be constructed by human beings. Instead, they happen to be there to determine human thinking and action. However, even if one is not a follower of the *Radical Constructivism* ((Maturana 1987), (Maturana and Varela 1987), (Glasersfeld 1995)), one may regard mental models as the result of a biological/chemical construction process that serves some biological purpose. Furthermore, we may regard the concept of mental model as a purposeful construction, an idea that is aimed at helping us with a better understanding of human cognition. According to the latter interpretation a mental model would not exist before somebody has developed the idea of such a construct – which in turn raises the Kantian question how one can develop an idea without having some corresponding concept, i. e. a mental model, already. Since specific characteristics of mental models as unconscious results of biological processes are not in the scope of our investigation (which does not mean that they are irrelevant to us!), we will refine the general concept of a model to that of a *model as an artefact*:

> A **model as an artefact** is a representation that results from a purposeful and conscious construction. It is perceivable and can be communicated. The abstractions it is based on depend on the purpose of the construction, mental activities of those involved in the construction and additional external stimuli and constraints.

Note that the idea of a given purpose that exists independently from a corresponding model is an intentional simplification. Usually, the meaning of a purpose will be shaped by a continuous act of interpretation that accompanies the act of modelling. From now on, we will use the term "model" in the sense of "explicit model". Note that the above definition gives no clues about the *quality* of a model. At the same time, we can assume that the benefit to be expected from a model will chiefly depend on its quality. Judging the quality of a model is far from trivial – especially, if one gives up the assumptions of naïve realism that recommend comparing a model against an original. Since the pivotal focus of this report is on conceptual models, we will get back to the issue of model quality after the introduction of conceptual models.

Characterizing a model as a construction serves to stress that it is not just a mapping of the "original" (if that exits anyway), but instead the result of a creative act which is influenced by cognitive abilities, expectations, language, technologies etc. Different from definitions that restrict the notion of a model to purposeful constructions, I also characterize them as an abstraction. With respect to a precise, clearly discriminating definition of "model" this may be regarded as a daring move since it seems to imply a precise definition of "abstraction" – which would be a hopeless undertaking. Nevertheless, it is possible to describe features of and intentions related to abstraction. At the same time, regarding abstraction as an essential characteristic adds meaning to the notion of a model as a construction. First, it allows for

distinguishing models from similar constructions such as photographs, realistic paintings etc. This corresponds to explicitly declaring a model as such[1]. Second, it underlines the need for deliberately going beyond the ostensible properties of the targeted domain. Usually, this act is related to fading out supposed properties that are not relevant for a given purpose. However, this is not sufficient. Often, abstraction also includes deliberately changing or adding properties to better serve the targeted purpose. The following examples illustrate this thought: Different from the original, the toy model of a car may include an electric motor and a remote control. Modelling persons may result in the abstraction "class of persons" with a property such as "average weight", which is not a property of a person, but a construction for a certain purpose. Hence, there is a mutual relationship between the two notions construction and abstraction. On the one hand, a construction is – in part – the result of an abstraction. On the other hand, the abstraction is a construction itself, in the sense that it creates something new. Therefore a model does not have to be a simplification of the considered part of a domain, nor does it have to represent a reduction of complexity at first sight. Instead, a model may even increase complexity by pointing at aspects that go beyond the obvious. However, as a consequence of making hidden, but nevertheless relevant aspects visible, such a model will promote the ability to successfully reduce and handle complexity. The fact that reduction of complexity often implies an increase of complexity first is well known from building tools: In order to build a tool, one has to increase complexity – by analysing requirements, resources, design options etc. Once the tool is implemented successfully, the complexity of the affected tasks will be lower. "Purposeful" is to express two aspects. On the one hand, it emphasizes that abstractions in this sense depend on conscious and intentional acts. On the other hand, it underlines – in the sense of Mahr's proposal – that a model needs to be deliberately created and introduced as such. This definition is rather generic (which nevertheless does not imply the claim to cover all reasonable uses of the term) and does not account for peculiarities of models in the area of Information Systems.

3 Information Systems

To further refine the conception of model for the field of Information Systems, we first focus on information systems which are a key subject of modelling in our field. Most definitions of information systems refer to "information" as a given term. While this will be sufficient in most cases, focussing on a concept of information first helps with gaining more clarity about the foundation and scope of information systems. Information is a key concept in many disciplines. We limit our investigation of the term to the realm of Information Systems.

[1] Note that a photograph or a "realistic" painting may include purposeful abstractions. But often, the idea will be to capture an image that clearly corresponds to the original.

3.1 Data, Information and Knowledge

A classical, often cited definition of information was suggested by Shannon, a mathematician and engineer (Shannon 1948). It is focused on determining the information transported in a message. The information content of a symbol within a message depends on the probability of its occurrence. While this formal information theory is of pivotal relevance for coding messages efficiently, it is of little use for the conception of information in Information Systems, because it completely fades out the meaning for prospective users. Developing a conception of information that is suited for Information Systems recommends discriminating it against conceptions of data. A plethora of definitions aims at the relationship between these terms. Others focus on the discrimination of information against knowledge. Especially in the latter case, the amount of diversity and terminological inconsistency is remarkable. Data is often regarded as a pure formal/technical representation. Information in contrast is created through an interpretation of data that creates meaning for humans (for corresponding definitions see for example (Laudon and Laudon 2005), p. 8, (Ferstl and Sinz 2005), p. 131). Information is sometimes seen as a precursor of knowledge that has to be somehow refined; hence knowledge is regarded as a special kind of information. In the German IS community many authors refer to a definition by (Wittmann 1959) who described information as purposeful knowledge, hence information as a special kind of knowledge (for example (Alpar, Alt et al. 2011), (Heinrich, Heinzl et al. 2010)). Slightly different from that, the authors of the FRISCO report suggest to regard information as "the knowledge increment brought about by a receiving action in a message transfer, i. e. it is the difference between the conceptions interpreted from a received message and the knowledge before the receiving action." ((Falkenberg, Hesse et al. 1998), p. 66). At the same time, they regard data as "any set of representations of knowledge" (ibid).

In accordance with many authors we regard data as symbolic representations. However, with respect to our specific focus, we do not regard any symbol as data. Symbols are core research subjects in various disciplines, especially in Semiotics. We regard "data" as a technical term of the Information Systems discipline. Due to the particularities of digitally represented data we suggest to restrict the term to digital representations that are stored in a computer or corresponding peripheral devices. However, with respect to the differentiated use of the term, such a conception remains too superficial. Therefore, we suggest refining it with respect to the purpose. There are two interrelated purposes data should serve. On the one hand, they are intended to represent something meaningful to prospective users, i. e. to represent information (see below). On the other hand, they should allow for being processed by a computer. Both aspects are related to the *semantics* of data. By semantics[2] we mean for-

[2] Note that semantics is sometimes – especially in Philosophy – regarded as a study or theory of meaning. Instead, we use the term according to Computer Science, where it is restricted to a formal interpretation.

mal semantics. The formal semantics of data is specified through a corresponding specification such as a *data type*. It defines the *extension* of possible occurrences (particular data of that type) and the formal operations that can be performed on them.

> **Data** are digital representations that allow for being stored on and processed by a digital computer. The structure and formal semantics of data is defined by a corresponding data type.

This definition may seem to be too restrictive since it does not include representations which are not assigned a type. However, sequences of bits are completely useless as long as they are not grouped into words that are assigned to types. The more restrictive a data type is, hence: the more possible interpretations it excludes, the higher the level of semantics it specifies.[3] As we shall see, this aspect is of pivotal relevance for information system integrity and integration. But even specifying data with data types is not sufficient for the efficient use of data: defining data as "Integer" or "String" still leaves an unacceptable wide range of possible interpretations. Only, if data is clearly related to the *information* it is supposed to represent, users have a chance to understand its *intentional semantics*[4], i. e. its *meaning*. This leads to an important consequence: Data without references to mental models of prospective users is meaningless. Different from pure formal rules of interpretation that constitute formal semantics, intentional semantics depends on the intension a human relates to the use of a term when communicating with others – which in turn depends on the language he speaks, i. e. the mental models, i. e. the cognitive concepts he uses, and his experiences (see (Habermas 1984), p. 311).

In general, information is created by symbolic representations – either discrete or continuous – that are suited to carry meaning for humans. Note that this does not mean that information is regarded as such by everybody – only by those who are able to relate it to a meaningful interpretation. Also, it does not mean that sender and receiver of information have to relate it to the same meaning (which to determine would require an adequate equivalence relation). The meaning of a representation is not embedded in the representation itself. Instead, it is created through references to objects or phenomena (such as behaviour, sentiments, action etc.), a user of this representation can perceive and corresponding concepts a user is familiar

---

[3] Note that in Computer Science, "semantics" is sometimes used as a Boolean predicate: There is semantics defined for an expression or it is not. We do not object to this use of the term. However, in accordance with a common practice in Computer Science, we also distinguish levels of semantics.

[4] The term "intentional semantics" is not used consistently. In Software Engineering, it is common to distinguish the extensional definition of a class (by defining the set of its instances) from an intentional definition (by defining the features shared by all its instances). Note, however, that an intentional definition in this case allows producing an extensional definition, too (by calculating the extension of a class from the extensions of the types of its attributes).

with. This is typically achieved by using assertions in a language the user speaks. For example:

"John Smith is a Student"

"John Smith has the matriculation no. IS-79281"

Data often do not represent assertions explicitly, but omit predicates like "has", "is a" etc. Nevertheless, a human observer may be able to associate data with corresponding assertions. A reference to an object without a corresponding concept does not – literally – make much sense. On the other hand, information may be restricted to references to concepts only. For example: "A customer is assigned zero to many invoices." Information is based on *difference*: An assertion is suited to carry information if it identifies specific cases out of wider range of possible cases. Hence, it defines a difference from possible constellations. In the realm of our investigation, information systems, we refine the concept of information by relating it to data. In other words: We focus on information that is represented through digital media. This specific conceptualisation of information is reflected in the following definition:

> **Information** evolves from data through an act of human interpretation. Interpretation is enabled by additional references to objects or phenomena and corresponding mental models (cognitive concepts). A reference can be established by strings that represent words of a language or graphical depictions. A reference may be limited to concepts.

The examples in Figure 1 illustrate the relationship of data and information. A table that contains digital data without any reference (example on the left hand side) hardly serves as information. Only if it is enriched with references (to the concept "Student" and to particular students), users may assign meaning to it and hence use it as information. Note that the representation on the right does not qualify as information per se. This is only the case, if it is observed by humans for whom the added designators work as adequate references. This conception of information corresponds to the semiotic rectangle.

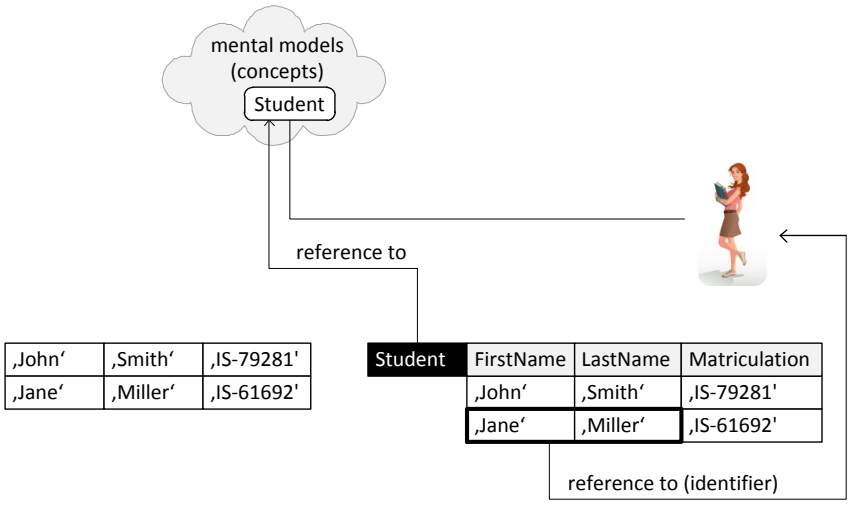

Figure 1: Illustration of creating information by enriching data with references to concepts and objects

This conception of information has an implication that is – as we shall see later – important for the conception of information systems: Information is not restricted to a certain representation on some medium. Instead, information is a cognitive construct that results from a construction that is triggered by references to cognitive models.

Note that the conception of information as a cognitive construction serves to emphasize the need for interpretation. Often, a relaxed use of the term "information" will be satisfactory: If data is represented with terms well known in a domain of discourse, we can refer to it as "information" directly. With respect to the utility of information, its content is important. The content of information relates to the difference from the expected. The more surprising information is, the higher its content. In other words: The more likely the interpretations are that are excluded by a piece of information, the higher its information content – and the more important its potential value for decision making. If an assertion represents a tautology, its information content is zero. The closer it gets to contradicting accepted assertions, the higher its information content.

The following examples illustrate this thought:

"Tom, the cat, is white."

"All cats are white."

"William White, a seasoned computer scientist, has never written a line of code."

"Martin Mellas, a lyric poet, has never written a line of code."

The first assertion excludes only Tom from being not white. The second excludes possible interpretations that are much more likely: that there are cats which are not white. Furthermore, the second assertion allows inferring the colour of a particular cat. At the same time, however, it is apparently not true. The third example comes as a surprise. For some, it will even contradict their conception of computer scientist. The fourth example, however, does not come as a surprise, but will probably correspond to many people's idea of a lyric poet.

Similar to "information" the term "knowledge" is not only used extensively in everyday speech but also largely overloaded. As already outlined above, there is no coherent use of the term "knowledge" in Information Systems either. It seems to be more suitable to regard knowledge as a special kind of information than the other way around: In the first case, knowledge would always be regarded as information. In the second case, there would be knowledge that does not qualify as information – which seems strange. We suggest a conception of knowledge that is inspired by scientific knowledge.

> **Knowledge** is characterized as information that represents a higher level of abstraction and satisfies the claim for justification. A higher level of abstraction means that it is not restricted to one particular case only, but covers a range of single cases.

The claim for justification reflects a rational assumption: Assertions, which are in principle refutable and that survived serious attempts to refute them, can – at least preliminary – be regarded as true.

The sentence “All cats are white” would qualify as knowledge with respect to the claim for abstraction. I would, however, fail to be supported by a convincing justification. If a justification is considered as convincing depends on the respective culture. As a consequence, the classification of information as knowledge may vary with the cultural surroundings. From an epistemological point of view, justification depends on various concepts of truth such as the correspondence, the coherence or the consensus theory (Künne 2003).

3.2 A Multi-Perspective Conception of Information Systems

There is a plethora of definitions of the term “information system”. The complexity of information systems makes it difficult to develop a definition that is both distinctive and comprehensive. At the same time, a discipline such as Information Systems requires an elaborate conception of its core research subject. To cope with the resulting challenge, we will suggest a conception of information system that makes use of various perspectives, which supplement one another. In a similar approach, Olivé proposes three perspectives on information system that are combined to a more comprehensive definition. The three perspectives relate to the contribution of information systems, their structure and behaviour and the functions they perform ((Olivé 2007), p. 1). The approach suggested here includes similar aspects, but adds further aspects that go clearly beyond Olivé’s definition. Note that we do not advocate a conception of an information system that excludes traditional media such as paper. Nevertheless, our main focus will be on computer-based systems. This is for two reasons: First, computer-based systems are already of pivotal relevance in most enterprises – and they can be expected to further penetrate organisations. Second, computer-based systems are not only of high complexity, but have specific peculiarities.

3.2.1 Focus on Purpose

Most definitions focus on the functions, an information system is supposed to provide. Typical examples are the conceptions suggested by Chaffey and Wood who define an information system as an instrument to "... gather, process, store, use and disseminate information" ((Chaffey and Wood 2005), p. 43) or by Bernus and Schmidt: “An information system is a system for collecting, processing, storing, retrieving, and distributing information within the enterprise and between the enterprise and its environment." ((Bernus and Schmidt 2006), p. 2)

To develop a more differentiated appreciation of the functions, an information system fulfils in organisation, it is useful to distinguish essential use contexts or perspectives. Laudon and Laudon distinguish three different kinds of information systems with respect to their purpose: “operational-level systems”, “management-level systems” and “strategic-level systems” ((Laudon and Laudon 2005), p. 43). While this wide-spread differentiation makes sense for analysing specific functions and associated data to be provided by an information system, it does not allow for distinguishing essentially different functions of an information

system in an organisation. The following differentiation is aimed at this purpose. It differentiates three principle perspectives on the functions of information systems: *information systems as instrument of information management*, as *organisational and management instrument* and as medium of *cross-organisational communication and collaboration*. Information management is aimed at improving organisational performance by supporting sourcing, creation, maintenance and preparation of information. This includes assigning information to tasks, caring for the appropriate, user-specific representation of information, providing for convenient access to information. In addition to that, information management has to account for information quality, confidentiality, trust, reliability etc. Last but not least, information management has to somehow assess the economics of information, i. e. relate the cost of information provision to its prospective benefit. An information system is a pivotal instrument for information management. On the one hand, it defines the type of data that is regarded as relevant in an organisation, on the other hand it is essential for making information accessible.

> An **information system** serves the effective support of prospective users by representing, storing, processing, retrieving and supplying data that serve its users as information. To provide information in a way that fits particular tasks and cognitive capabilities, it may allow for adapting the presentation of data to specific tasks and/or roles. To fulfil these purposes, an information system has to support data integrity and to provide mechanisms for protecting data.

To promote the appropriate and efficient handling of information it is also required to assist users with developing corresponding competencies and attitudes, e.g. with respect to information awareness, information sharing etc. An information system may include features that also support these people-oriented tasks of information management.

Some authors propose a conception of information system that explicitly includes human actors as carriers and transmitters of information (e.g. (Schwarzer and Krcmar 2004)). With respect to the outstanding relevance of information available through humans only, such a wide conception makes sense. In other words: It would be naïve assuming that all information that is relevant in organizations is explicitly stored on some media. Nevertheless, we prefer a narrower conception of the term that is focused on digital, computer-based systems, but does not entirely exclude the use of traditional, e.g. paper-based, media. This is for two reasons: First, from an Information Systems perspective we regard information systems as artefacts that are designed and maintained using software engineering methods. This focus is for a good reason: The complexity of information systems as artefacts is remarkable and requires a specific expertise. However, an engineering perspective is not appropriate for humans as potential parts of information systems. Second, information created and provided by humans suggests a different perspective that accounts for facilitators and inhibitors of information sharing, such as interest and power, the function of images, metaphors, storytelling and gossip. Note that preferring a narrow conception of information system does not mean to deny the relevance of human actors for providing, disseminating and using information in organisations. Furthermore, an information system is not an end in itself. Its con-

ception always requires accounting for the organisational action system it is supposed to support, i. e. for human action and (cross-organisational) collaboration. This aspect is emphasized in the following definition.

> An **information system** represents rules for performing activities such as processes and decisions. This includes the definition of tasks, processes, roles, obligations and access rights. Therefore, an information system is an instrument for organizing the firm.

A similar definition is proposed in the FRISCO report where an information system is regarded as "a sub-system of an organisational system, comprising the conception of how the communication- and information–oriented aspects of an organisation are composed" ((Falkenberg, Hesse et al. 1998), p. 72 f.).

Intra-organisational communication and collaboration is more and more based on the use of information systems. Customers or other external actors may access a company's information system directly to retrieve information or to start transactions. Collaboration between organisations can be further promoted by establishing cross-organisational workflows. Interactions between an organisation and its environment may thus be more and more represented by interactions between corresponding information systems.

> An **information system** may serve as a medium to enable communication and collaboration with external actors, thus contributing to the realisation of *inter-organisational systems*.

This aspect emphasises the need for enabling information systems to communication across organisational border, i.e. to integrate them (see 3.2.3).

3.2.2 Focus on the Artefact

The previous definitions focussed on functions and effects, thus widely avoiding a description of what an information system consists of. Including this aspect in our investigation recommends adding a further perspective on information systems that concerns their essential structure or – in other words: the "substance" they are made of:

> An **information system** is a *linguistic construction*. It consists of two principle layers: a schema and corresponding instances. The schema defines the formal semantics of the instances managed by an information system. The schema is implemented with languages that can be translated to the instruction set of a processor without the loss of semantics. A schema can be an instance of a further, higher-level schema.

At first sight, the above viewpoint may be regarded as trivial: Information systems are implemented by some kind of implementation language. However, emphasizing the linguistic foundation of an information system means more than that. The semantics of implementation languages are based on abstractions of machines, i. e. the data representations and instruction sets provided by micro processors. For an information system to serve the purposes discussed above, this kind of languages is not sufficient. This is for two reasons. On the one hand, constructing an information system from basic data types and generic functions only

will usually imply an unacceptable effort. Instead, there is need for concepts on a higher level of semantics, preferable directly related to a certain domain, that can be used for constructing an information system. On the other hand, describing an information system with domain-level concepts is required to give users a comprehensible image of the system. Note that the concepts used for documentation purposes do not have to correspond directly to the concepts used for constructing the system. Nevertheless, these concepts are constitutive for the perception and – literally – conceptualisation of an information system. The schema may also include implementations of operations on these data. Note that it is possible – and sometimes beneficial – that an information system has more than two layers of abstraction. The higher the level of semantics featured by the schema, the better are the chances to keep data in a consistent state, i. e. the more possible interpretations are excluded, the better are the chances to prevent illegitimate states. The following examples illustrate this thought: If customer revenues are specified as floating point numbers, negative values are possible. To exclude this threat to integrity, one could restrict the range of possible values – either through an appropriate data type or through an additional integrity constraint. If the purchase price of a product must never be larger than the retail price, this could be enforced by a corresponding constraint. If data that represents a particular aspect of the respective domain is stored more than once, we speak of data redundancy. Data redundancy causes an additional effort for updating data, which results in a threat to data integrity.

From a traditional engineering perspective, an information system consists of linguistic artefacts such as application systems, components or objects. The behaviour of an information system results from the interaction of the artefacts it is made of.

> An **information system** is composed of artefacts on different levels of detail and abstraction. The composition of artefacts is based on interfaces which represent commonalities.

Distinguishing different levels of abstraction is essential for the construction of information systems. In particular, the dichotomy of instances and types (or classes respectively) is characteristic for most information systems, since it is implied by prevalent programming languages. Note that we will use the terms “class” and “type” widely synonymously in this report. That does not mean that we regard them as equivalent. Both in Logic and in programming languages, there are clear differences between the two terms. However, for our consideration it is sufficient to not further distinguish between types and classes. A type defines the properties that are characteristic for a set of corresponding instances. Thus, it allows for changing an instance population according to the constraints defined with their common type. The differentiation of types and instances corresponds to the differentiation of schema and occurrences (instantiations), which is often a constituent part of information system architectures. Note that “type” and “instance” represent a role of an object within an instantiation relationship. A type may be regarded as an instance of a meta type (see 4.7). A schema defines structural and maybe functional and dynamic properties or corresponding instantia-

tions. Hence, it allows for describing a system on a higher level of abstraction without account for specific states of the system.

The engineering perspective reminds of the construction of machines or buildings. Similar to these, it also demands for an architecture.

> An **information system architecture** is an abstraction of an information system. Hence, it is a *model*, which represents the *structure* of a system, i. e. how it is built from functional units and how these interact. Different from conceptual models used in system analysis and design domain-specific particularities are usually faded out. An information system may be represented by different architectures.

The conceptualisation of interfaces as "plugs" is useful to some extent only. A deeper understanding of information systems requires a more elaborate conception of interface. Unlike a "plug", an interface may have to cover a wide range of concrete constellations that correspond to an elaborate, abstract schema. Regarding interfaces as contracts enables a further perspective on information systems that stresses aspects that are not only relevant from an engineering point of view.

> **Information systems** are constituted by patterns of contractual interactions. Each participating artefact needs to comply with a contract that defines its responsibilities. A contract can be more or less rigorous.

While some design methods, e.g. "design by contract" (Meyer 1997), demand for making contracts explicit, contracts are often used implicitly, if they are used at all. However, there are explicit or implicit *constraints*, which can be regarded as contractual assertions, too. They comprise e.g. features of classes or pre- or postconditions. Integrity demands for restrictive contracts, while flexibility demands for less-restrictive contracts. Coping with this conflict constitutes one of the major challenges of information systems research (see below).

3.2.3 Focus on Integration and Reuse

There are two further aspects that are essential for information systems design: *integration* and *reusability*. They account for the fact that information systems are often complex artefacts that include various components, such as software modules or application systems. While integration has been a key design objective for information systems for long, the term "integration" is often used without introducing a specific concept of integration. Generic definitions of integration as they can be found in dictionaries usually emphasize the composition of a whole from initially separate parts. For developing a differentiated appreciation of integration in Information Systems, such a definition is not of much help because it does not account for the peculiarities of IT artefacts. Integrating two IT artefacts, e.g. two components or two application systems, is aimed at enabling them to collaborate. Collaboration implies a common subject. In a general sense, two components collaborate on a common subject, if they can both access and modify it. Hence, one component is affected by the other component modifying the common subject. Since information systems are linguistic artefacts, ac-

cessing common subjects requires communication, which in turn implies common concepts or a common language. We also speak of a common *semantic reference system*.

Communication between IT components can be differentiated with respect to the subject, which can be static (data), functional (functions, services) or dynamic (processes). Furthermore, we can distinguish direct and indirect communication. If the primary subject of communication is data, then two components can either directly exchange data, e.g. by passing messages, or indirectly, e.g. by changing data in a common data store. In both cases, the participating components need to have a common concept of the shared data. If communication is focussed on functions, direct communication implies that one component calls a function or service provided by the other component. In the case of indirect communication, a public function is used by both components to access and modify common data. In this sense read access to a public function, e.g. to a time service, would not enable integration. Integration that makes use of common functions allows for a higher level of abstraction. This effect, referred to as information hiding or encapsulation, is of crucial importance for system flexibility. Using indirect communication instead of direct communication also enables a higher level of abstraction by abstracting from the receiver of a message. Finally, integration on a dynamic level means that respective components are enabled to collaboratively execute a process. In the case of direct communication, a component would respond to a notification, e.g. an event, emitted by another component by continuing the common process. Indirect communication would require a central control unit that receives all relevant events and triggers the subsequent function. Functional integration implies static integration (common concepts to describe data) and dynamic integration requires functional integration (common functions to be called for performing a process).

Table 1 gives an overview of the various kinds of communication that may constitute the integration of IT artefacts.

	static	functional	dynamic
common concepts	types/classes (e.g. Integer, Real, Customer …)	functions/services/ methods, e.g. getCustomer (id: String) …	process schema (e.g. order management workflow), events (e.g. “invoice created”)
direct	inter-component messaging passing data	calling function /service/method provided by other component	Component responds directly to event created by other component.
indirect	accessing and modifying common data	calling shared function that modifies common data	Central control unit receives events and triggers subsequent function according to common process schema.

Table 1: Different kinds of Integration

While common concepts, such as data types, function declarations or event types are prerequisites for integration, their existence does not guarantee a satisfactory quality of integration.

The quality – or level – of integration depends on the quality of the respective communication. In general, the quality of (machine) communication depends on the semantics of the common concepts: The higher the level of semantics, i. e. the more possible interpretations are excluded, the higher is the quality of communication (with respect to a certain purpose). For instance: If two components exchange data as instances of the type "Real", it may represent anything, thus creating interpretation/transformation effort and jeopardizing system integrity. If they instead use a common concept of customer revenues the range of interpretations would be clearly smaller – as well as the threat to integrity. Figure 2 illustrates this conception of static integration.

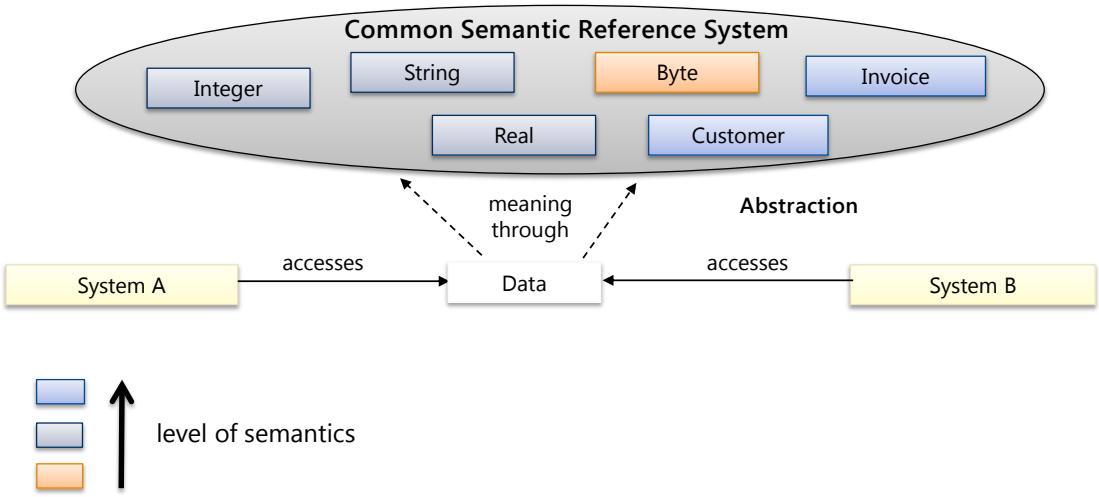

Figure 2: Levels of Static Integration

Figure 3 serves to illustrate the same idea for dynamic integration by distinguishing event types with respect to their semantics. Often, workflow management systems do not have access to application-specific events such as "order.sum > limit". Instead, they are restricted to events occurring outside of application systems, e.g. those produced by the file system. As a consequence of this low level of dynamic integration, the chances to adequately control the execution of a process would be limited.

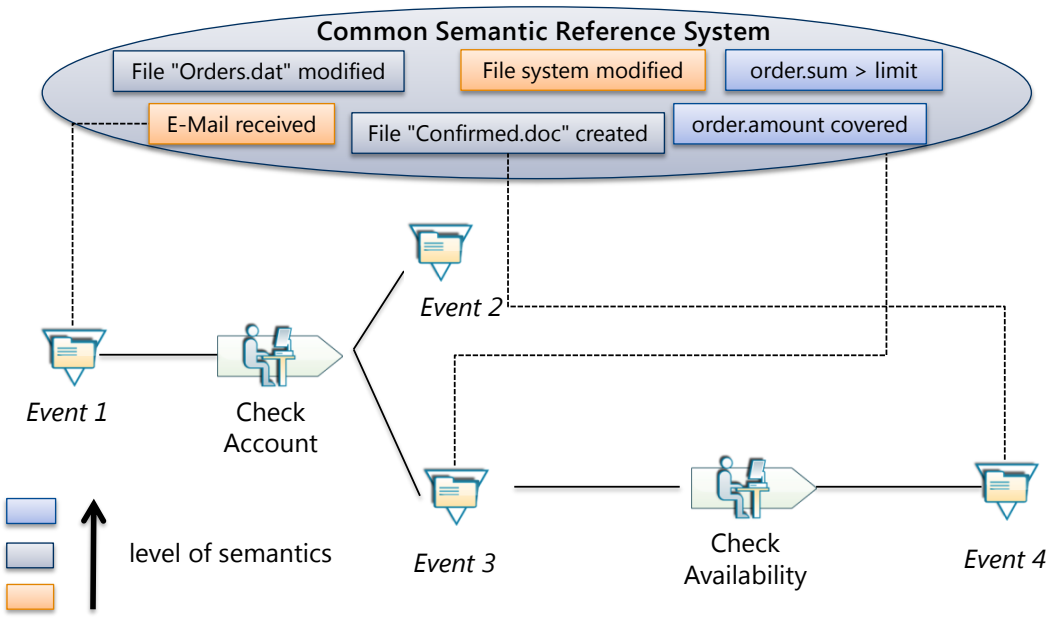

Figure 3: Levels of dynamic integration

Communication requires common concepts that are specified in a common semantic reference system such as a class schema. To properly use these common concepts there is need for a *common namespace* to refer to them in an unambiguous way. In addition to common concepts, integration may also comprise a population of common instances that can be directly referred to using a common namespace for instances. Figure 4 shows a common namespace for jointly managing instances. The semantics of instances is defined by a reference to the corresponding concept in the semantic reference system.

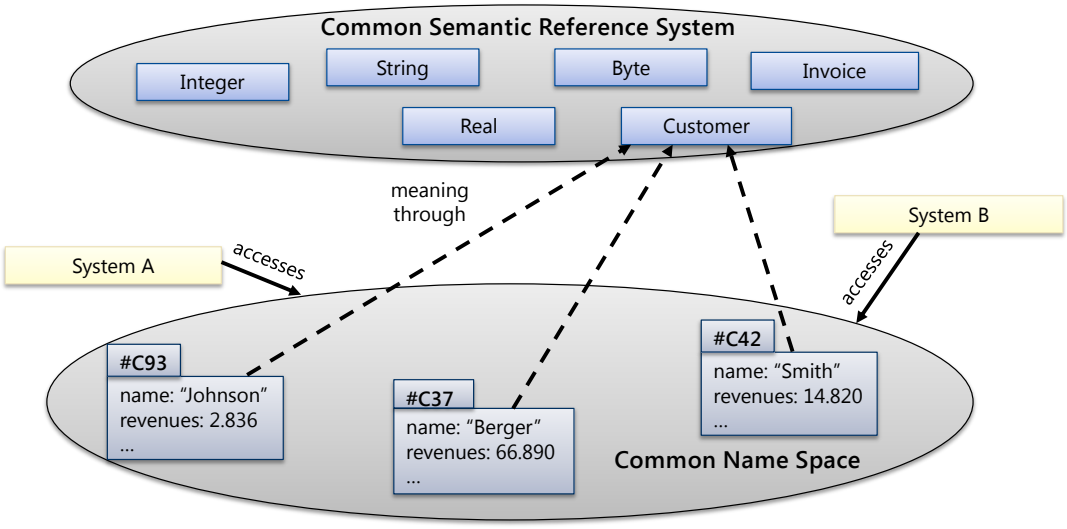

Figure 4: Illustration of instance-level integration

Instance-level integration is a prerequisite for avoiding redundancy, hence, a promoter of integrity. Establishing common instance namespaces is a particular challenge in distributed systems with limited bandwidths.

Reusability is the other side of the coin. There is no reuse of artefacts without integrating them into existing environments. Also, reusing an artefact within a range of contexts requires these contexts to share common requirements. Reuse is a major driver of productivity in traditional manufacturing. It allows accomplishing higher quality at a lower price. Reuse is even more attractive for information systems: Once the artefact is implemented, the actual production costs are almost zero resulting in tremendous economies of scale. For a more elaborate study on the prerequisites of integration and reusability in information systems design see (Frank 2008). The effect of semantics on integration and reuse is ambivalent. The following propositions, illustrate this problem:

- A high level of integration promotes effectiveness (no need for reconstructing semantics) and integrity.
- Integrating components on a low level ("loose coupling") facilities exchanging components with less effort and contributes to flexibility.
- Semantics promotes the benefit of reuse.
- Semantics compromises the range of reuse – and therefore prevents attractive economies of scale.

It is one of the key challenges of Information Systems research to overcome or relax the above design conflicts. This requires developing abstractions that cover a wide range of potential use cases and allow for convenient and safe adaptations.

Integration concerns also the relationship between an information system and the action system it is supposed to support. In this case, we speak of *organisational integration*, often referred to as "IT-business alignment". It is similar to the integration of data or components in the respect that it is based on linguistic considerations. At the same time, it is different, because the language used to specify the concepts an information system is based on is clearly different from the natural language that is used in the organisational universe of discourse. Therefore, organisational integration cannot aim at common concepts in a strict sense. Instead, it aims at a correspondence of concepts in the information system with those in the respective universe of discourse. The better the match between a concept in the universe of discourse and the corresponding concept in the information system, the higher is the organisational integration. Matching is not just a matter of structural or operational similarity, but depends also on designators. If, for instance, the concept of a customer in an information system is represented to a user through a respective designator of a language he does not understand, there will be friction, i. e. integration will be dissatisfactory.

> **Organisational integration**: The level of organisational integration is the higher, the more the concepts of in information system and the designators used to represent them resemble the terms in the organisational universe of discourse.

For instance: If an information system provides users with domain-independent concepts such as "file", "directory", "table" etc., the user needs to link these concepts to the technical terms he is familiar with, such as "Sales Report", "Product Directory" etc. The bigger the

semantic gap between a concept represented in the information system and a concept used in the corresponding action system, the higher is the chance for friction (misunderstanding, reconstruction effort, inconsistencies).

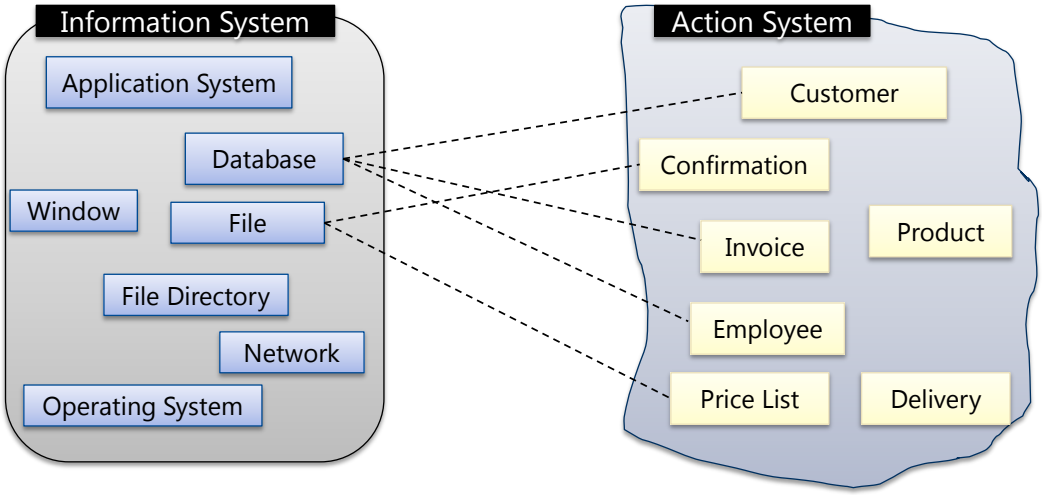

Figure 5: Example of poor organisational integration

Figure 5 shows an example of poor organisational integration. In addition to conceptual aspects, organisational integration concerns the *synchronisation* of an information system and the corresponding action system. Whenever a relevant aspect of the domain has changed, the information system should be updated accordingly. At the same time, changes that occur in the information system should be accounted for in the domain.

> The state of an **information system** should be synchronised with the corresponding state of the represented domain.

Recent developments in the area of transponder systems enable an inspiring vision: Physical objects that carry a rich description of themselves may in the end – provided current bandwith bottlenecks can be overcome – make it obsolete to distinguish between physical objects and their representation in an information system.

In addition to the synchronization of instance-level states, one could also consider the evolution of an organisation and the schema of its information system. It relates to what is often called flexibility or adaptability of an information system. If a domain gets rearranged in a way that effects information system requirements, then the schema of an information system needs to be adapted. In other words: The evolution of the relevant universe of discourse and the schema of an information system should be synchronized. This kind of synchronization is especially demanding because, change of this kind may require mutual adaptation.

3.2.4 Images of Information Systems

Information systems are complex artefacts that have a substantial impact on the way organisations work and how they are perceived. Therefore, people need to develop an understanding of information systems. Since information systems are invisible artefacts, they are difficult to grasp. Therefore, images or metaphors may play an important, however, ambivalent role. On the one hand, they may contribute to all too simplified and therefore misleading conceptions of information systems. On the other hand, they may help to create simplified conceptions of information systems that foster an appropriate understanding from a certain point of view. In his seminal book on "images of organizations", Gareth Morgan (Morgan 1986) suggests to take advantage of the expressive power of images and metaphors. He proposes to use multiple images of organisations to promote a multi-perspective conceptualisation of organisations and, hence, a deeper understanding. His proposition is based on the premise "that our theories and explanations of organizational life are based on metaphors that lead us to see and understand organizations in distinctive yet partial ways." ((Morgan 1986), p. 12)

Information systems are different from organisations in various aspects: They are purposefully constructed artefacts, they are mere linguistic constructions, the tasks they perform follow certain algorithms, etc. However, at the same time they also resemble organisations in important aspects: Organisations are complex, too. They are, to a large extent, based on social, hence, linguistic constructions. Organisational actions follow more or less rigidly prescribed goals and rules. For people working in organisations it is important to develop an appropriate understanding, not only of the organisation, but also of the information system. Against this background, metaphors and images may be also beneficial for gaining an appropriate understanding of information systems. In the following, we will look at four possible images, each of which focuses on a particular aspect of information systems. They serve as examples for how to promote the appreciations of information systems from various standpoints.

Regarding information systems as a *strategic weapon* focusses on an attractive perspective that may be enabled by advanced information systems: An information system can be used to gain competitive advantage, e.g. by reducing costs, providing additional services etc. This image of information and information systems has been proposed by various authors (e.g. (Porter and Millar 1985), (Mata, Fuerst et al. 1995)). While this image may be regarded as too martial by some, it can be useful for those who are in charge of strategic planning – of the firm in general and the information system in particular: It stresses that it can pay of to regard the information system as a key enabler of sustainable differentiation. That includes the conception of new business processes, the development of new products as well as entirely new business models. However, using this image in a naïve way is not appropriate: It is not just technology that needs to be implemented. For an information system to promote long-

term differentiation, it has to be co-designed with a company's action system, its products, its relations to external partners etc.

Looking at information systems as a *commodity* emphasizes a substantially different view. This image that has been advocated most notably by (Brynjolfsson 1993) and (Carr 2004) who challenged the presumed positive effect of information technology on competitiveness. Carr uses analogies to other technologies such as railways or electricity to propose that any new technology provides opportunities only in an early phase of its existence to later become a commodity that is accessible by everybody with similar costs. This view of information systems as a commodity has been criticized by many for good reasons. However, in recent years it has gained growing popularity again through a slightly different image, which is not related to the idea that information technology does not matter. Regarding information technology as a *service* is similar to the commodity image in the sense that it suggests convenient availability on demand without the need to bother with realisation and maintenance. Using this image as a purposeful abstraction may help with getting a clearer view of how the information system should support the business. It may also help with defining separation of concerns to promote professional management of information systems. On the other hand, this metaphor has its downsides, too. Often, it is misused by suggesting that the realisation of complex information systems is just a matter of picking the appropriate services (which already exist) and "orchestrating" them to support the business. Usually, the remaining challenges such as data persistence are not mentioned in these images.

Information systems may promote the image of "organisations as machines" ((Morgan 1986), p. 19): The software determines processes, does not allow for exceptions and does not account for personal relationships. In this sense it contributes to the conception of bureaucratic organisation as it was proposed and analysed by Max Weber. As with bureaucracy, the corresponding effects of information systems are ambivalent: In those cases where the rules embedded in the information system are appropriate, they foster productivity and consistency – and reduce complexity for users. However, in cases where unforeseen requirements may occur, the machine metaphor can contribute to inhibiting change.

Information systems may also result in "psychic prisons": "Human beings have a knack of being trapped in webs of their own creation." ((Morgan 1986), p. 199) The functions information systems provide, the input they request and the results they deliver constitute to a large degree what people perceive as organisational reality: It determines the way they perceive the business and hinders them to think beyond existing work patterns. This effect may be regarded as problematic, since it is an inhibitor of change or may even be used as a subtle instrument of domination (as it was suggested by Habermas or Marcuse in their assessment of technology on society in general). In any case, this image comes with the demand not to take information systems and the patterns of their use for granted. Instead, a reflective user should once in a while think about how information technology is influencing his action and

organisational collaboration – in order to imagine new ways of conceptualizing and using information systems.

From a more optimistic perspective, information systems can also be regarded as promoters of "self-organisation" or, as Morgan puts it, as "brains" – even though not everybody will like this kind of mystification. If information systems are based on powerful abstractions that do not only cover the world as it is, but also a wide range of possible future worlds, they will not compromise change, but instead act as enablers of change. This is even more the case with information systems that integrate representations of their surroundings and possible goals they are supposed to aim at. For a corresponding conception of information systems see (Frank and Strecker 2009).

4 Conceptual Models and Modelling Languages

An information system is a linguistic artefact which is based on the concepts that constitute software systems, e.g. classes, modules, methods etc. It is aimed at representing relevant aspects of a certain domain. At first sight, it may appear that a system developer analyses the respective part of reality to then represent it using appropriate technical terms from the field of software engineering. However, a closer look shows that nobody will directly look at reality (whatever that is). Instead, one will talk to those who are familiar with the domain or will read descriptions of the domain. Hence, analysing a domain requires analysing existing linguistic representations of the domain. The term "domain of discourse" – sometimes referred to as "universe of discourse" – serves to express this fact.

> A **domain of discourse** is constituted by the language that is used in a certain domain. It is characterized by specific concepts that reflect the topics relevant for performing meaningful actions in that domain. Often, these concepts are not precisely defined. Instead, their interpretation – which is embedded in *language games* – may vary within the domain.

4.1 Conceptual Model

The focus on linguistic constructions leads to the notion of a *conceptual model*. Conceptual models have been at the core of information systems analysis and design for long. An often cited definition of conceptual models stresses mainly the second aspect: "... descriptions of a world enterprise/slice of reality which correspond directly and naturally to our own conceptualisations of the object of these descriptions." ((Mylopoulos and Levesque 1984), p. 11) While models that correspond directly to conceptualisations their prospective users are familiar with will certainly promote comprehensibility, restricting conceptual models to this aspect is misleading: It will often not be the only aim of a conceptual model to provide a "natural" representation that corresponds to existing conceptualisations in the targeted do-

main. This is for two reasons: (a) the unavoidable mismatch between natural language and implementation languages and (b) the need to overcome limitations of current practice.

Ad (a): A conceptual model in Information Systems is usually aimed at bridging the gap between concepts used in a domain and the concepts used for implementing a system. There are various differences between natural language and prevalent implementation languages. In contrast to natural language, implementation languages require formalisation. Many natural language concepts resist against formalisation because understanding their meaning relies on comprehension and empathy (e.g. (Wright 1974), p. 20). Furthermore, the semantics of key abstraction concepts in prevalent implementation languages is different from corresponding concepts in natural language. For instance: In natural language, an instance may be an element of many classes. In (most) object-oriented programming languages, an object is an instance of one and only one class. This difference between an extensional and an intentional conception of class has serious but subtle implications on the semantics of specialisation (see e.g. ((Frank 2000b; Frank 2000a) . To cope with this problem, it will often – not always – make sense adapting the semantics of a modelling language concept to that of respective implementation languages in order to avoid a mismatch that jeopardizes a straightforward and consistent transformation.

Ad (b): As outlined already above, a conceptual model may be aimed at a representation of a possible world which is intended to provide an orientation for change. In other words: The idea of a possible world is thought to overcome existing shapes of a domain. At first sight, it may appear that such an objective is not in conflict with Mylopoulos' and Levesque's definition: One would describe the future world in terms of the language people in the respective domain are familiar with. However, a closer look reveals that this would be an inappropriate belief – which requires refining our preliminary definition of a model. While we speak of modelling a domain, we actually do not directly focus on actions or physical objects within the domain. Instead, we will usually regard a domain of discourse, hence viewing the domain through the lens of a given language, i. e. the language of the domain experts who describe the domain to us. This thought reflects a precondition of recognition that has been known since Kant: "Also ist die Erkenntnis eines jeden, wenigstens des menschlichen Verstands eine Erkenntnis durch Begriffe, nicht intuitiv, sondern diskursiv." ((Kant 1976), B 92, 93).[5] Hence, a conceptual model is an abstraction of an existing linguistic abstraction. In other words: It is the result of a *twofold abstraction*. Again, one might assume that this consideration is of philosophical relevance only. But this is definitely not the case. First, it emphasizes the pivotal relevance of language and communication, if we want to picture a domain. In their seminal work on the "social construction of reality" Berger and Luckmann emphasize as insistently as convincingly the crucial role of language for what they call the objectivation of

[5] „Hence, the recognition of any, at least of the human reason is recognition through concepts – not intuitively, but discursively."

the world, which is constituted by living with and through the language we share with others (Berger and Luckmann 1966). The late Wittgenstein stresses this aspect, too, by introducing the term "language game": "Here the term "language-game" is meant to bring into prominence the fact that the *speaking* of a language is part of an activity, or of a life-form." ((Wittgenstein 2001), §23). With respect to our focus, the analysis and design of information systems, these considerations emphasize the pivotal role of language – both as an enabler and inhibitor of abstraction: "Language is my instrument – but simultaneously my problem, too." (translated from (Maturana 1987), p. 90 f.) – or as the early Wittgenstein put it: "The limit of my language means the limit of my world." ((Wittgenstein 1981), §5.6).

Against this background, I suggest a conception of "conceptual model" that is rather extensive:

A **conceptual model** is a model that is characterized by the following features:

Linguistic construction: A conceptual model is created through the use of a *modelling language*. This does not only exclude physical models, but also models that are depicted in natural language or some unspecified graphical notation.

Abstraction from particular instances: As a default, a conceptual model does not represent particular objects. Instead, it is built of concepts which are abstractions of objects of the same kind. This is an important prerequisite for protecting a model against the threat of ever changing instances – and for applying (reusing) it to other instance populations. Note that there are exceptions to this rule.

Independence from technological change: Conceptual models are supposed to abstract from confusing and ever changing peculiarities of technical infrastructures such as implementation languages, operating systems and hardware. Thereby they contribute to the protection of investment – into the development of models and corresponding systems.

Correspondence to spoken language: While there are various ways to build concepts, conceptual modelling is aimed at making use of concepts that are part of the language spoken in the targeted domain.

Integrative abstraction: While there are good reasons to focus on the domain of discourse only and to abstract from technical issues, using a conceptual model for designing software will eventually require mapping them to implementation-level representations. Hence, a conceptual model is not only an abstraction of the targeted domain, but at the same time an abstraction of the corresponding information system. To support the mapping of a conceptual model to implementation-level representations, it is important to account for the semantics of the targeted representations in advance (see ad (a) above).

Reconstruction of concepts: While conceptual models are built on analysing terms of a corresponding domain of discourse, their concepts are not necessarily aimed at representing existing concepts. Instead, it may be required to clarify and even adapt the meaning of concepts with respect to the purpose of the conceptual model. In other words: Designing a conceptual model does not mean to merely represent relevant concepts of a certain domain of discourse, but *reconstructing* them.

Emphasizing the reconstruction of concepts is for two reasons: First, the concepts used in conceptual models will often serve to bridge the gap between a domain of discourse and an information system. Therefore, they will in part reflect the limitations that are characteristic for implementation languages (see previous aspect). Second, conceptual models may be used as instruments for promoting change. Therefore, they may include concepts that deliberately deviate from existing concepts to overcome their limitations.

The reconstruction of concepts from the corresponding domain of discourse serves two interrelated purposes. On the one hand, it supports the development of adequate linguistic structures, such as the structure used to represent the concept "Customer". On the other hand, it adds semantics to a conceptual model that goes beyond the formal semantics defined with the corresponding modelling language. This is accomplished by representing concepts with designators that serve as references to terms of the corresponding universe of discourse. Figure 6 illustrates the importance of terminological references for the appropriate interpretation of a conceptual model. With respect to their syntax and formal semantics the three models are equivalent. However, it is very likely that an observer will have no idea what model a) means, because it does not contain any reference to known terms. This is different with models b) and c). Since they include references to terms we are familiar with, we have a clearly better chance to interpret them. Model b) will be regarded as inappropriate by most observers, because many invoices can be sent to a customer, while an invoice will usually be assigned to exactly one customer. Therefore, model c) will be the only one that makes sense to most of us.

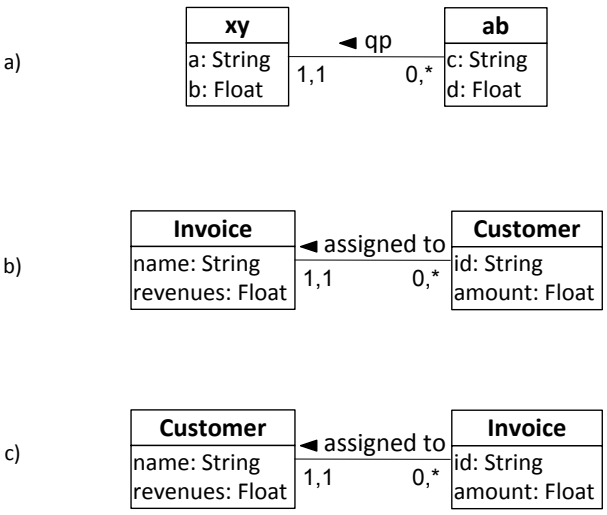

Figure 6: Impact of references to domain of discourse for meaning of a conceptual model

The example illustrates that a conceptual model can hardly be reduced to its formal semantics. Instead, its utility depends chiefly on associating it with conceptualisations the observer is familiar with. This marks a clear difference to models in formal logics. There, a model is an interpretation of a formal expression that makes this expression true (Wolters 1984). For in-

stance: The natural numbers can be regarded as a model of the Peano axioms. Despite this difference, there is a similarity: The designators used in a conceptual model can be seen as references to mental models of observers. A mental model can be regarded as a framework of interpretation. Only, if one – out of many – mental model is associated that seems to fit, the corresponding conceptual model will make sense to the observer (which does not mean, however, that it has to be regarded as "true" or "appropriate"). Therefore, it is misleading to regard a model as a "clear, precise and unambiguous conception" ((Falkenberg, Hesse et al. 1998), p. 55). Note that a reference can also be established by an iconographic symbol.

This thought relates to the so called *semiotic triangle*. Figure 7 shows an adaption of the semiotic triangle that corresponds to core terms of our previous considerations. Note that the figure covers selected aspects and relationships only. While it suggests that mental models are just there, we can assume that they are influenced by interaction and communication, hence, by domains of discourse and also by conceptual models. At the same time, conceptual models are not determined by corresponding domains of discourse, but may also contribute to changing related natural languages. This consideration is of especial relevance for the construction and use of information systems: It illustrates that the language we use (domain of discourse) serves as a foundation of the information systems we design (through corresponding conceptual models) and that the information systems we use may change the language we speak.

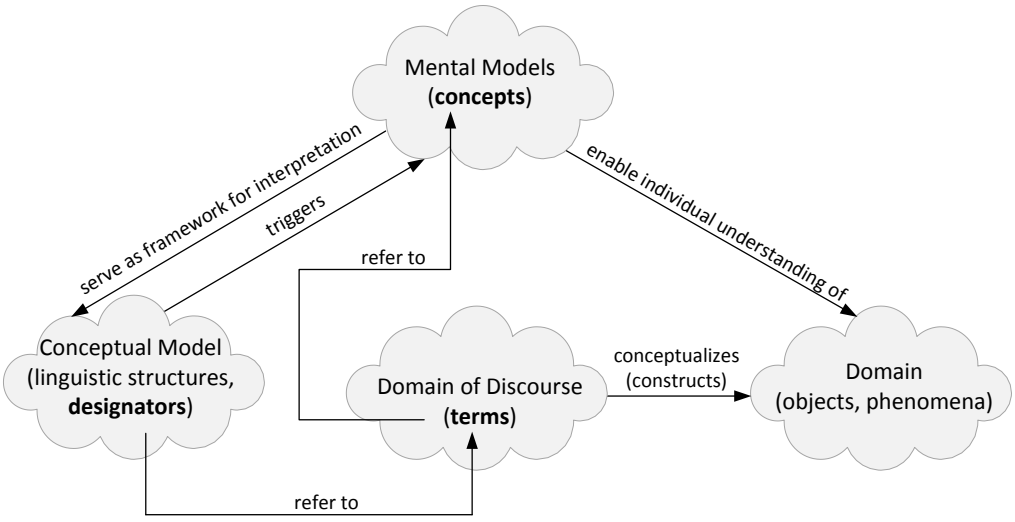

Figure 7: Semiotic rectangle, adapted from semiotic triangle

4.2 General Purpose and Domain-Specific Modelling Languages

Conceptual models require the use of a *modelling language*.

> A **modelling language** is a language that serves to create a class of conceptual models (from now on referred to as "models"). For this purpose it provides a set of concepts. A conceptual modelling language is defined through its syntax and semantics. The *ab-*

stract syntax defines rules for constructing syntactically correct models using the language concepts. The *concrete syntax* defines the symbols used to represent the abstract syntax. Since these symbols are usually graphical, it is also referred to as graphical notation. The *semantics* of a modelling language defines the (formal) interpretation of modelling concepts. It can be based on a formal specification. Often, the semantics of a modelling language is not completely formalized.

Note that there is no clear difference between the abstract syntax and the semantics of a modelling language. The formal semantics of a concept is – among other things – defined through its associations with other concepts. Possible associations can be specified through the abstract syntax or through semantic constraints (for an example see (Frank 2010), figure 16). While a modelling language is pivotal for the development of conceptual models, it can be more or less suited for a certain range of modelling tasks. Among other things, this implies the question how much domain-specific semantics a modelling language should include. This leads us to the distinction of domain-specific modelling languages (DSML) and general purpose modelling languages (GPML).

A **GPML** is a modelling language that is thought to be independent from a particular domain of discourse. Instead, it should be suited to cover a wide range of domains. It consists of generic modelling concepts that do not include any specific aspects of a particular domain of discourse.

A GPML provides its users with basic concepts such as "entity type", "class", "attribute" etc. which are used to reconstruct the relevant terms of the respective domain of discourse. At the same time, there are mappings of the basic concepts to concepts required for the design of information systems. Figure 8 illustrates how a conceptual model is created from basic concepts and can be mapped to foundational concepts of a corresponding software system.

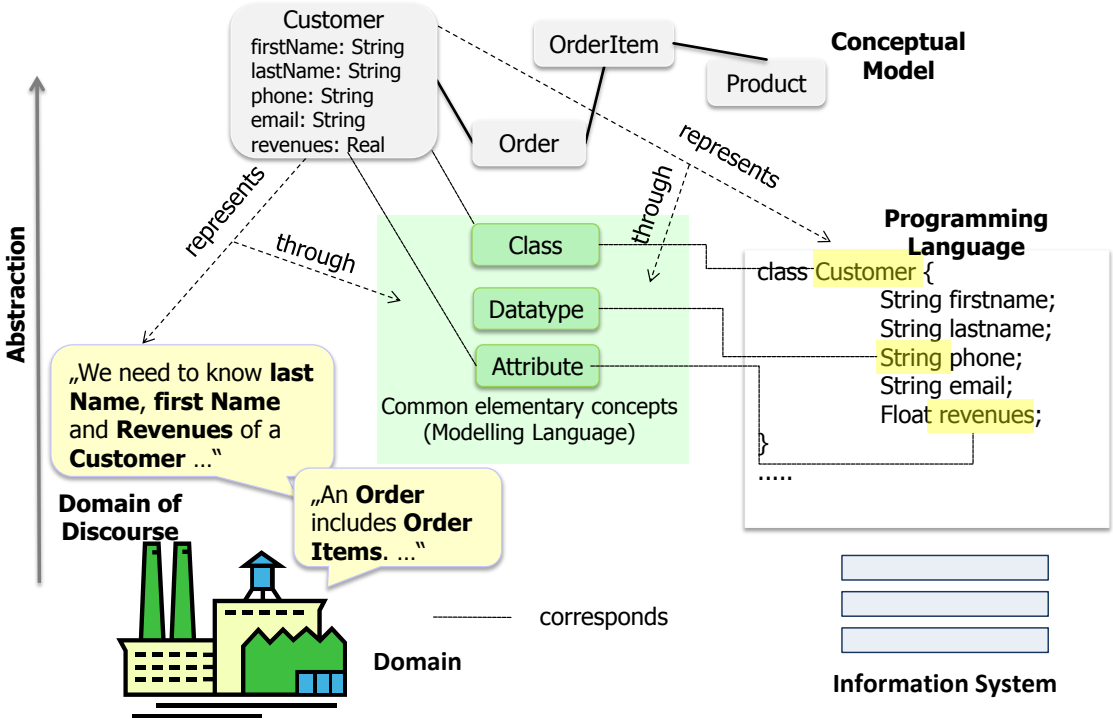

Figure 8: Illustration of mapping from conceptual model to code through GPML

Even though created with a different motivation, the basic concepts that form the foundation of GPML correspond to those found in the philosophical study of Ontology. Especially the categories proposed by Grossmann and Bunge show a clear correspondence to the concepts used in GPML. Grossmann proposes categories such as "individuals", "properties", "relations", "classes", "quantifiers"(Grossmann 1983). Bunge, who speaks of the "furniture of the world", suggests that the world can be described with basic concepts as "things", "properties of things", "attributes of things", "classes of things", "laws and lawful states", and "coupling of things"(Bunge 1977).

Many system designers will regard a GPML as a self-evident instrument the use of which they consider as beneficial. However, in everyday life we would regard it as an entirely unreasonable demand to restrict our communication to a language with a few primitive concepts only. Instead, we expect a language to provide concepts that allow for differentiated communication without forcing us to explain everything from scratch. In recent years, this thought has led to the development of modelling languages that were designed for specific domains:

> A **DSML** is a modelling language that is intended to be used in a certain domain of discourse. It enriches generic modelling concepts with concepts that were reconstructed from technical terms used in the respective domain of discourse. A DSML serves to create conceptual models of the domain it is related to.

Using a DSML releases modellers of the need to reconstruct domain-level concepts from primitive concepts. As a consequence, one can expect a higher productivity and a contribution to integrity, because the DSML excludes inconsistent models more effectively than a GPML (for an illustrative example see (Frank 2011), figure 1). In recent years, the same thought that has motivated the development of DSML has produced domain-specific programming languages (DSPL). They provide programmers with domain-specific concepts, which increase productivity and foster software integrity. In an ideal case, models specified in a DSML are mapped to code of a corresponding DSPL (see example in Figure 9). It is also conceivable that both, code and model are based on the same representation.

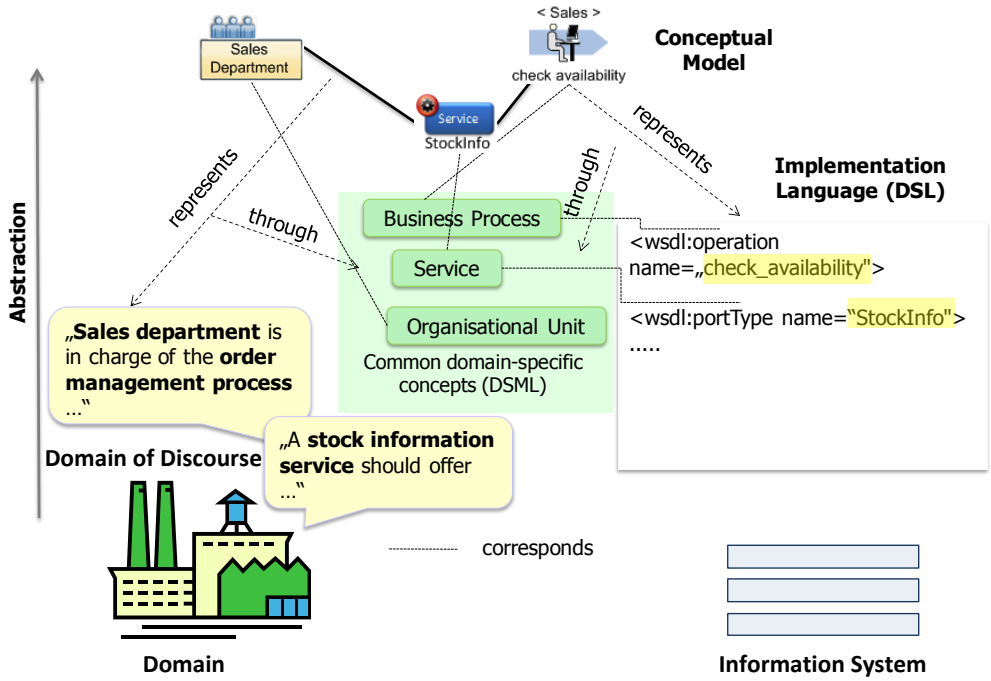

Figure 9: Illustration of mapping DSML to DSPL

4.3 Abstraction Concepts

Modelling goes along with abstraction. Advanced abstraction does not only mean to leave out aspects of the represented subject ("projection"). Instead, it is related to coping with variety and change.

4.3.1 General Considerations

The extension and the state of "things" that are covered by a model may change over time. Furthermore, the requirements related to a system may change over time or within the intended range of its (re-) use, which would result in the need to modify/adapt the system. Hence, for a particular system the appropriate level of abstraction depends on the subject, the design purpose and assumptions about future changes. In any case abstraction should not be a reflection of ignorance: Leaving out aspects simply because we feel incompetent to analyse them is a daring idea. Instead, abstraction should be based on a thorough and well-grounded analysis of the domain and the system to design. For example: An architect who designs a building has to decide what aspects he can/should abstract from in order to promote the building's adaptability to future changes. He may consider aspects such as interior, wiring, garden design, lighting conditions, windows, economy or security. To decide whether it is a good idea to abstract from a particular aspect, two criteria are of special relevance:

a) the dependency of the system to be built from the aspect and

b) the knowledge about possible realizations of the aspect.

Ad a): The more the design of the system depends on a certain aspect, the less are the chances to abstract from this aspect without jeopardizing the entire construction. The structure of a house will usually be not independent from the windows. On the other hand, the design of a building will usually not depend on the prospective interior. Ad b): It makes sense to abstract from aspects the occurrences of which may vary in time. However, if one knows only little about possible realizations, it may turn out that the design of the building is not independent from some possible future realisations. Wiring, for instance, may have been done in a certain way over decades. But then new requirements, e.g. related to digital networks, and new technological options may suggest accounting for wiring when designing the structure of a house.

The brief example shows that abstraction should be aimed at descriptions that cover the range of potential states and potential modifications of the targeted system. That means that all respective states and modifications correspond to permissible interpretations of the model. The concept of a class is a typical example for covering a range of possible system states: A class is defined by a set of properties such as attributes, methods or associated classes. Each attribute in turn is specified by a type or a class, which represent a clearly defined set of possible occurrences. From a formal point of view, all combinations of these sets of occurrences would define the range of system states covered by a model. With respect to system integrity, a respective abstraction should cover all conceivable system states, but exclude impermissible (or insane) states.

More advanced abstraction is aimed at promoting the adaptability of a system. In addition to the current system, it also covers possible future adaptations of the system. As a consequence, adapting a system to new requirements means to concretize an abstraction, i. e. to replace/extend the abstraction with a more concrete specification. A corresponding abstraction should cover all possible future modifications. At the same time, it should widely exclude modifications that would result in impermissible (or insane) systems. For conceptual models to represent abstraction, corresponding language concepts are required. A thorough analysis of abstraction concepts goes beyond the scope of this report. We will restrict our analysis to an overview of prevalent abstraction concepts and a more detailed consideration of generalisation/specialisation and classification/instantiation.

4.3.2 Prevalent Abstraction Concepts

In addition to abstraction concepts provided by modelling languages, there are other forms of representing abstractions, e.g. through patterns or aspects. However, we will not consider these here. On the one hand, they represent a variety of different, sometimes not clearly specified abstractions. On the other hand, they are not provided by object-oriented modelling languages and respective programming languages, which are a relevant reference for our study, since it accounts for the transformation of models to code. While many object-oriented modelling languages do not include delegation, it is accounted for nevertheless because it is an important alternative to generalisation. In any case, abstraction is aimed at coping with

complexity that is caused by sometimes subtle diversity. Abstraction concepts allow for purposefully abstracting from diversity – of system states and possible future versions of a system. Thus, they may allow for a more comprehensible representation of a system. At the same time, they support procedures to change the state of a system or to create a new version without jeopardizing its integrity. Table 2 gives an overview of abstraction concepts. Note that the definitions given for classification and generalisation are preliminary only.

Concept	Description	Abstraction from
Classification	Serves to classify a set of congenerous objects, i. e. assign them to a common class. All properties defined for the class, apply to each of the included objects. For example: Objects representing customers can be classified into a class "Customer" that specifies all properties they share. The opposite direction of classification is called *instantiation* and means to create an object according to the specification of a certain class. Example: An object to represent a new customer is created from an existing class "Customer".	the concrete extension of a class, i. e. its actual set of objects and the states of the objects of a class
Encapsulation	Also referred to as "information hiding". It means that the internal structure and state of an object are invisible from the outside, i. e. the state is not directly accessible. Instead access to an object's state is restricted to the access paths (methods) the object provides with its interface.	the internal structure and state of an object
Generalisation	results in a superclass that represents the common properties of a set of subclasses. A superclass represents the set of properties, a set of subclasses have in common. Every proposition that is true for the superclass is true for its subclasses, too. Example: "Master Thesis", "Sales Report" etc. are generalized to "Document". The opposite process, i. e. creating a subclass, is referred to as *specialisation*. It describes a relationship between two classes. A class that is specialised from a superclass extends the set of properties of the superclass by further properties. Example: The superclass "Document" is specialized into "Master Thesis" by adding the properties "title" and "author". Specialisation implies the *substitutability constraint*: Any instance of a class can be substituted for an instance of one of its subclasses.	from the peculiarities of existing and possible future subclasses
Delegation	represents a directed association between a role object and a role filler object. For certain assignments, the role filler object delegates its responsibility to a role object. The role object refers (transparently) to its role filler's properties and state.	from the specific responsibilities of existing and possible future roles
Polymorphism	allows for resolving the semantics of a message at run-time only. In other words: A message with a particular name may relate to multiple methods (implementations).	from the class of the object a message is sent to

Table 2: Overview of Abstraction Concepts

To analyse the benefit of abstraction concepts it is useful to distinguish between their effect on managing a particular system within a certain time frame where it is stable and on supporting the evolution of a system. In the first case, the focus is on managing the changing instance populations of the system and providing users with a comprehensible system representation. In the second case, the focus is on supporting convenient and safe (consistent) modifications.

Concept	Particular System	System Evolution
Classification	*Comprehensibility*: Classes that correspond to concepts in the respective domain of discourse, foster ease of use and organisational integration respectively (by reducing friction between information system and action system). *Integrity*: The states of objects can be changed within the boundaries defined with the corresponding class. New objects of a class can be created with no need to specify their semantics again. The more restrictive the implicit and explicit integrity constraints specified with a class, the higher the contribution to system integrity.	Extending classes by adding new properties (attributes, methods) is monotonic and does not corrupt previous responsibilities of a class.
Encapsulation	*Comprehensibility*: To understand the responsibilities of a class, it can be sufficient to look at is interface only. In this case, encapsulation reduces complexity.	The internal structure of an object can be changed without causing side-effects, as long as the object interface does not change.
Generalisation	*Comprehensibility*: If the classes of a generalisation hierarchy correspond to concept generalisation known in the respective domain of discourse, the system may become more comprehensible. However, there are two obstacles to account for. On the one hand, extensive generalisation hierarchies may be confusing. On the other hand, the semantics of generalisation in programming languages is usually different from that in natural language – which may cause confusion.	Adding subclasses enables *reusing* the properties of the superclass and allows for system adaptations that are not a threat to integrity – adding a further subclass represents a *monotonic* extension.
Delegation	*Comprehensibility*: The concept of a "role" is widely used. Therefore, it should foster the comprehensibility of a system's representation. *Integrity*: Since many role objects may	Adding new roles to a role filler allows for reusing the role filler's properties. Also, adding new roles is monotonic and does not threaten existing responsibilities.

	refer to one corresponding role filler object – including its state (!), delegation contributes to reducing redundancy and, hence, to promoting integrity.	
Polymorphism	*Comprehensibility*: Using the same designator (as a well-known abstraction) for a set of similar methods is suited to foster comprehensibility.	Adding more specific subclasses that overwrite methods of the superclass does not require changing the classes of those objects that are supposed to use these new methods.

Table 3: Benefits of Abstraction Concepts

Note that the particular meaning of generalisation – as well as the need for delegation – depends on the concept of a class, which may vary.

Differentiating between a given system and its evolution corresponds to common views: In the first case, the focus is on managing instances, in the second case, the focus is on managing schema modifications. But the boundaries between these two views may be blurred. Usually, they are related to peculiarities of prevalent implementation languages: Most languages are restricted to two levels of abstraction (type and instance). While changes to instances can be applied during run-time, schema modifications often require re-compilation. However, there are cases where schema modifications are required during the run-time of a system. For example: A text editor may require adding new document types during run-time. An online-retail platform may require adding new product types during run-time. Furthermore it is noticeable that the selected abstraction concepts represent static and in part functional abstractions only.

4.3.3 Appropriate Abstraction: Some Postulates

The benefits of abstraction concepts as they are outlined in Table 2 relate to consistently coping with diversity and change. However, not any kind of abstraction is of the same quality. The following postulates serve to outline essential aspects of model quality. The more they are accounted for, the higher is – ceteris paribus – the quality of a model. Postulate 1 and 2 relate to models of a particular system, while postulate 3 and 4 relate to models that are supposed to cover the evolution of systems.

Postulate 1: *Conceptual fit*. This postulate addresses the appropriateness of abstractions. On the one hand, it implies that a model includes all concepts that are relevant and each concept includes all relevant properties, where relevant means "required for serving the purpose of a system". On the other hand, conceptual fit refers to parsimony: No more concepts – and for each concept no more properties – than those that are relevant should be part of a model.

Postulate 2: *Integrity*. This postulate is related to the previous one. In particular it refers to the semantics specified for the concepts of a model. The semantics of a concept is defined by the set of possible occurrences (instances) and the set of possible operations on these occurrences. To promote integrity, a concept should allow for representing all relevant occurrences

and operations. For example: The analysis of a sales department reveals that every sales representative is assigned exactly one sales district. Specifying this restriction in a corresponding model implies that a respective system would not allow for assigning more than one district to a sales representative. However, it may turn out that there are cases where it makes sense that a sales representative is assigned more than one district. As a consequence, this may either restrict the use of the system for this purpose or compromise its integrity, since users may look for a "workaround". At the same time, integrity demands for inhibiting the representation of impossible/nonsensical instances. If, for instance, a property that represents the revenues of a customer is specified as String, it will be possible to assign values that do not represent numbers. At the same time, the semantics of a String would not allow for relevant operations (see above). Usually, this demand is addressed by using predefined data types. As a consequence, it will usually be fulfilled to a small degree only, since data types often allow for instances that are nonsensical. For example: Specifying revenues as Real would not only allow for negative values but also for numbers that are bigger than any reasonable value. The integrity of a system is especially an issue when its state is changed. A functional abstraction, e.g. a method provided by a class, should cover all relevant transitions – and prevent those transitions that compromise system integrity. If a relevant transition is missing, users may again apply some sort of workaround and thereby jeopardize system integrity. A potentially inconsistent transition would for instance be a function that allows for computing negative amounts in stock. The legitimacy of a transition may depend on previous transitions, i. e. it may be path-dependent. In this case, a dynamic abstraction such as a state chart or a process model is required. A dynamic abstraction should cover all relevant paths of execution – and exclude those paths that would result in inconsistent system states.

While it is demanding enough to satisfy the above postulates for one particular system, it will usually be not sufficient, because system requirements tend to change. Abstraction becomes much more challenging when we look at the evolution of a system – or its differentiation in a set of variants. It means that postulates 1 and 2 should be satisfied not only for a particular system, but for an entire range of system variants or versions that evolve over time. In general, coping with change or variety recommends distinguishing between those parts of a system that are invariant and those parts that may vary. At first, this demand constitutes an epistemological challenge, since it requires making reasonably reliable assumptions about possible variations and future changes. Of course, these assumptions can be shattered: Action systems are contingent and subject of unforeseen change, i. e. people are creative in inventing new technologies and in establishing new patterns of action. Apart from this epistemological challenge, a further challenge remains: Even, if we are confident to some extent that we can distinguish between system parts that are invariant within a certain time frame and those that are not, there is still need to adequately represent this knowledge in a model. As a general rule, the invariant parts of a system should never be deleted. The variant parts may be deleted or replaced. Hence, the invariant core of a system is subject of reuse

through the range of its variants and the evolution of future versions. An ideal abstraction is focussed on invariant parts of a system. At the same time, it abstracts from the range of variations that can be accomplished by extending the invariant part in a monotonic way, i. e. without jeopardizing the correctness of the invariant part. For example: A class "Document" may be an invariant part of a document management system. Further document types may be subject of future changes. They can be specialized from "Document" without having an effect on the invariant core. For an abstraction to foster convenient and consistent system modifications, it should satisfy the following postulates:

Postulate 3: *Monotonic extension*. Adding new features to a concept that is represented by the abstraction should be monotonic. Specialisation that satisfies the substitutability constraint is a prototypical example for a monotonic extension.

Postulate 4: *Variability*. An abstraction should allow for concretions that cover the entire range of relevant variants or versions. The following example illustrates how postulate 4 may be satisfied or violated. If all product types within an inventory management system and its variants/versions are characterized by corresponding instances that have a certain size, weight and identification number, all future product types may be added as specialisations. However, if product types such as certain liquids are added that do not have these features, they could not be introduced through specialisation.

Postulate 5: *Evolutionary integrity*. An abstraction should not allow for concretions that result in nonsensical systems, i. e. systems that one does not want to see at all. Specialisation, for instance, allows for an unlimited number of possible extensions, most of which will usually not make sense. In an ideal case, satisfying this postulate would enable systems that allow advanced users to apply modifications without threatening system integrity.

To satisfy postulate 4 and 5 one would need to know all possible variants and future versions of a system and be able to determine which variants and future versions have to be avoided. Apart from this – already mentioned – epistemological challenge developing abstractions that satisfy the above postulates is possible to a limited extent only. Monotonic extensions are restricted to static and functional abstractions only. Dynamic abstractions that aim at representing processes within a system do not allow for monotonic extensions (see (Frank 2012). Therefore, separating invariant from variant parts of a dynamic abstraction is an essential challenge of system evolution.

4.4 Diagram and Diagram Type

Often, the terms "conceptual model" and "diagram" are not clearly distinguished. Sometimes, this is not a problem. However, with respect to designing modelling languages and building corresponding tools, there is need for a more elaborate terminology.

A **diagram** is a visual representation of data – both on the instance level and on conceptual levels. A diagram can be regarded as a view on a model or on a set of interre-

lated models. It can comprise an entire model (or set of models) or only a part of it. Different from conceptual models, diagrams will often represent selected features of instance populations. Hence, every (graphical) model is represented by a diagram, but not every diagram represents a conceptual model.

An example of a diagram is the graphical representation of a data model. An enterprise model that has been created using multiple modelling languages could be represented in one multi-language diagram.

A **diagram type** represents a class of congenerous diagrams.

The notion of a diagram type is important for describing modelling methods, since it allows for describing views on models that are appropriate for certain tasks or scenarios.

4.5 Ontology

In Philosophy, the term “Ontology” is established for the study of the “being”. It is aimed at identifying what is “real”, analysing the relevance of experiences for this purpose and at finding basic categories of being that allow to describe all forms of being (for an overview see (Schwemmer 1984). Apart from the philosophical tradition, the term “ontology” is also widely used in Computer Science and Information Systems. This is an unfortunate situation, because it contributes to terminological confusion. This is even more the case as both uses of the term have commonalities. In addition to that the use of the term in Computer Science and Information System is not consistent either. Often, it is used without an explicit definition. If a definition is given, it usually refers to Gruber who introduced the term in the field of Artificial Intelligence as "an explicit specification of a conceptualization." ((Gruber 1992), p. 1) Even though Gruber explicitly refers to the philosophical term “Ontology”, he does not bother much with a closer look. Instead he neglects the relevant relationship between Ontology and Epistemology and states that – in the realm of knowledge-based systems – "what 'exists' (in the sense of philosophical Ontology, U.F.) is exactly that which can be represented.” ((Gruber 1992), p. 1) Subsequently, the term “ontology” gained remarkable popularity, although the definition is hardly convincing: Every data model constructed with the Entity Relationship Model would qualify as an ontology. Unfortunately, the term is often not used with a conception that would allow a clear distinction from “conceptual model”. Some authors compare them to models on the object level, while others relate them to meta models ((Henderson-Sellers 2011), p. 100 f.) – resulting in a remarkable terminological confusion. However, while the intention that is related to ontologies shows clear similarities to conceptual models, there are a few characteristic differences in the actual construction of ontologies and conceptual models. The most important difference relates to the specification language. In the case of ontologies, specification languages (e.g. OWL or RDF) allow for deduction, while this is usually not the case for languages used to specify conceptual models. Related to that, ontology specification languages are usually not specified with meta models, but with grammars. Also, different from the default for conceptual models, an ontology may cover

various levels of abstraction, e.g. combine the object level with a meta level. In addition to that, conceptual models are usually expressed through graphical diagrams, while ontologies are usually represented as text.

> An **ontology** is a conceptual model specified in a formal language that allows for deduction and usually lacks a graphical representation. It may cover various levels of abstraction.

Apart from the unfortunate use of the term, the deduction capabilities offered by ontologies can be a substantial benefit. They allow for inferring characteristics of objects from other objects by applying general inference procedures such as backward chaining. Unfortunately, corresponding languages are based on a paradigm different from prevalent object-oriented languages, which makes enhancing object-oriented languages with deduction capabilities a serious challenge.

4.6 Reference Models

DSML support the development of conceptual models by providing domain-specific concepts for reuse. A similar objective is addressed by *reference models*.

> A **reference model** is a conceptual model which is intended to represent a domain that comprises a class of possible applications, e.g. all companies of a certain industry. Reference models come both with a descriptive and a prescriptive claim. On the one hand, they should account for actual patterns of action and corresponding concepts. On the other hand, they are supposed to serve as a blueprint for especially effective information systems.

The latter aspect is stressed by the claim that they should be accepted as a reference by a wider range of users. While this claim is directly related to the essential objective of reference models – to achieve advantages through economies of scale – it is at the same time problematic. This is for various reasons: First, it is a criterion that is not related to features of a model itself, but depends on its dissemination. Second, it is unclear what level of acceptance – both in range and commitment – is required to speak of a reference model. Third, if taken seriously, the reference claim would widely compromise the use of the term "reference model" in IS research, because so far there are only very few models that – arguably – can be regarded as an accepted reference. Despite these obstacles, I do not object to the use of the term. It is an established term in IS research and – more important – it represents an attractive long term vision for research which to pursue makes sense even if the intended dissemination cannot be achieved. In addition to foster productivity through reuse, reference models promote (cross-organisational) integration of those information systems that are based on them. In this case, the concepts defined in a reference model serve as a *common semantic reference system* for all systems to be integrated.

While reuse alone would allow for adapting a reference model to individual needs, adaptations are a threat to maintenance and integration. Hence, the more diverse the domain – and

the concepts in the corresponding domain of discourse are, the more challenging is the creation of reference models. If the diversity of key concepts such as "customer", "product" etc. is too high, reference models reach their limits. A type in a reference model can be modified only by deleting or adding – arbitrary – properties. Allowing for a wider range of adaptations requires a higher level of abstraction. This is the case for DSML: Different from a reference model on the object level, the concepts a DSML is comprised of are defined in a meta model, hence, they are usually abstractions of types. In other words: A DSML may be intended to serve as a reference model – on a higher level of abstraction. The following example illustrates why DSML are suited to cover a wider range of concept diversity than reference models on the object level. An e-commerce platform is supposed to offer a wide range of entirely different products such as furniture, consumer electronics, vehicles etc.

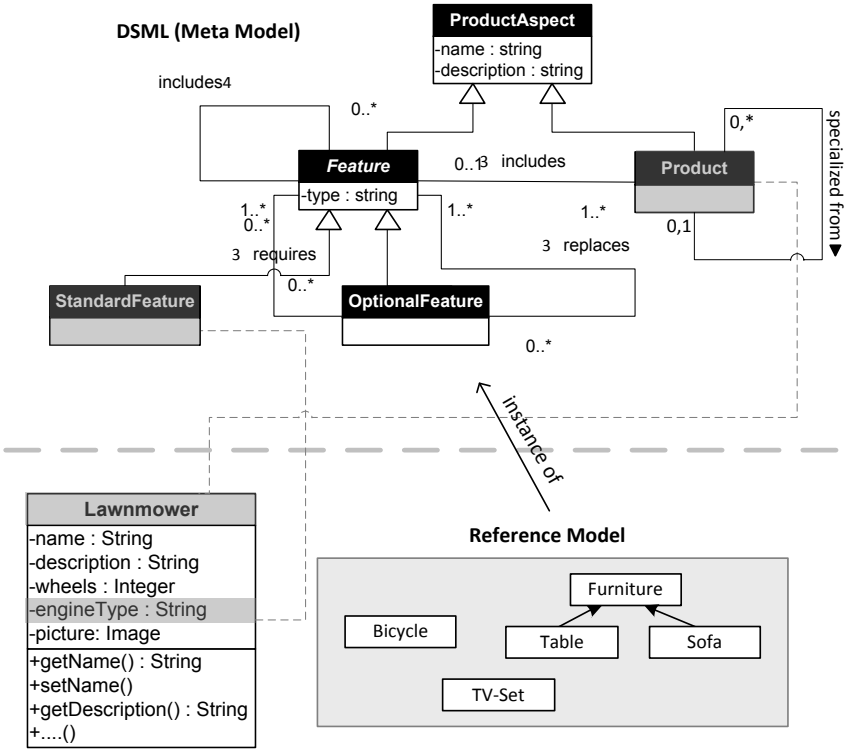

Figure 10: Combining advantages of reference models and DSML

Representing all these products in a reference model would result in numerous product types. Nevertheless, it would not guarantee to cover all product types that may appear in the future. Also, it would not guarantee to adapt the model to new product types by specializing them from existing ones, since extending the set of features may not be sufficient (for a detailed analysis of this problem see (Frank 2001)). One possible solution would be to choose a flat concept of product that would, e.g., be reduced to name, price, description etc. This would, however, compromise the corresponding systems' functionality, e.g. with respect to supporting elaborate search procedures, and their integrity. Using a DSML that includes a generic concept of product would support the consistent and elaborate representation of particular product types. Compared to a reference model, a DSML provides increased flexibility,

but still requires specifications. To take advantage of the convenience offered by reference models and the flexibility of DSML, both approaches can be combined. Figure 10 shows an example. Note that it does not include a specific graphical notation of the DSML.

In the long run, DSML enable an attractive vision: All relevant technical languages could be reconstructed through DSML. If they were standardized, they would then serve as a global "lingua franca" for representing information systems and corresponding IT artefacts. To foster the integration of DSML, they should be specified with a common meta language and share common concepts. The prospects of this vision are remarkable. While it may be regarded as frightening by some, it would also dramatically improve the efficiency of cross-organisational collaboration and boost economies of scale through reuse.

4.7 Meta Classes and Meta Models

The specification of a class implies a linguistic structure that comprises the concept of class. A concept of class, i. e. the specification of all possible classes of a certain kind, can be defined with meta classes.

> A **meta class** is a class of congenerous classes. Since a meta class is a class, there may be an indefinite hierarchy of meta classes.

So far, we used the term "class" and "type" in most contexts without further differentiation. With respect to the specification of modelling languages, a differentiation between meta classes and meta types makes sense – even though the definition of a meta type corresponds directly to that of a meta class:

> A **meta type** is a type of congenerous types. Since a meta type is a type, there may be an indefinite hierarchy of meta types.

The abstract syntax and semantics of a modelling language can be specified by a grammar or a *meta model*. Meta models are especially appealing for this purpose for two reasons. First, they are based on the same paradigm as the models themselves. Hence, prospective language designers should be more familiar with them than with grammars. Second, they provide a better foundation for designing corresponding modelling tools than grammars, because they can be transformed to object models in a straightforward way. Meta models are widely used in Software Engineering and Information Systems. Nevertheless, there are still debates about the conception of meta models.

> A **meta model** is a conceptual model of a class of conceptual models. Hence, a meta model determines how to specify permissible models of a certain class. In other words: It defines the abstract syntax and semantics of a *modelling language*. As a default, the key concepts of meta models are meta types. There are, however, exceptions to this rule.

A meta model could be built of meta classes, too. However, if a meta model serves the specification of a language independent of a machine, the definition of class methods does not make much sense. Since a meta model is a conceptual model itself, an indefinite hierarchy of

meta models is conceivable. Since a meta model is supposed to specify the concepts of a class of models, the concepts a meta model consists of, should be more elementary. If the concepts of a meta model are regarded to be clear enough, e.g. if there is a formal foundation they are based on, there is no need for introducing a further meta layer. Often, but not necessarily, this sequence is limited to meta meta models. It has become popular to use numbers for identifying model layers, where zero corresponds to the instance layer, one to the type layer, two to the meta type layer etc. Even though such a structure can be very helpful to distinguish different layers of abstraction, it has its limitations.

4.8 Method and Modelling Method

While a modelling language is mandatory for developing a conceptual model, it is not sufficient. Instead, the complexity of the tasks will usually demand for a professional approach or, in other words, for an appropriate *method*.

> A **method** is aimed at solving a class of problems. It consists of a terminology, proven assumptions about successful action and a corresponding process model that guides the course of problem solving steps.

Typical examples of methods are diagnostic procedures used by physicians: They are based on the technical language used in medical science, which provides a linguistic structure of the problem, medical knowledge (assumptions about symptoms and related diseases) and process models that guide the course of a diagnosis.

> A **modelling method** is a specific kind of method. It is aimed at solving a class of problems through the design and use of models. It consists of at least one modelling language and at least one corresponding process model which guides the construction and analysis of models. In addition to that it includes assumptions about successful patterns of action.

Note that often the term “modelling methodology” is used instead. This is, however, misleading: A methodology is a study of methods.

> A **domain-specific modelling method** makes use of at least one DSML and at least one domain-specific process model.

We call a modelling method that is aimed at supporting the design of meta models or the design of modelling languages a meta modelling method:

> A **meta modelling method** is a modelling method that consists of a meta modelling language and a corresponding process model which guides the specification of meta models, which in turn may serve the specification of a modelling language.

A meta modelling method is different from a regular modelling method in two respects. On the one hand, it requires the use of a meta modelling language that serves specifying the intended modelling language. On the other hand, it demands for specific attention towards the analysis of corresponding requirements: Often, prospective users do not know what to ex-

pect from an artefact they have not come in touch with before. (Frank 2010) presents an approach that accounts for this challenge. Furthermore, there are modelling methods that are aimed at the construction of modelling methods. Corresponding activities are often referred to as "method engineering". Often, their main focus is on the design of process models, which is supported by specific abstractions for designing process models. These abstractions may include a language for specifying process models as well as reference process models. (for a recent overview of approaches to method engineering see (Henderson-Sellers 2010)). In addition to that method engineering is usually aimed at realising modelling tools to facilitate the efficient use of a corresponding modelling method.

Method engineering is the discipline of systematically designing modelling methods.

In its most basic form it comprises a modelling language and a corresponding process model for designing process models for the use of certain existing modelling languages. More advanced forms of method engineering include the modification/extension of existing language specifications or even the creation of new languages. They require a respective meta modelling method. To foster the efficient realisation of modelling tools, meta modelling tools are of great value, since they enable the automatic creation of rudimentary modelling tools out of a language specification.

5 Enterprise Modelling

Conceptual modelling is aimed at the development of application systems in general. In any case, enterprise modelling has a more specific and at the same time a wider scope. It is based on the assumption that enterprises are domains with specific peculiarities and remarkable complexity. Therefore, they require a dedicated modelling approach. In addition to supporting the development of enterprise software systems, enterprise modelling is also aimed at supporting analysis, design and management of the enterprise itself. Therefore, the definition of the term "enterprise model" recommends to first consider essential characteristics of enterprises.

5.1 Conceptions of the Enterprise

There is a remarkable range of approaches to define the term "enterprise". Some suggest conceptualizing enterprises as social systems, or as social-technical systems. Many other focus on core objectives, such as generating profits or maximizing profits. The "inducement-contribution-theory" is aimed at explaining why individuals would join an enterprise and why enterprises can be stable systems over a longer period ((Barnard 1938), (Simon 1949)). Further approaches emphasize a specific economic perspective. In his theory of the firm, Gutenberg proposes mathematical models of idealized production processes to analyse the use and combination of scarce resources (Gutenberg 1929). The transaction cost approach (Williamson 1985) targets the question, if and when institutions such as enterprises are supe-

rior to markets where individual actors would offer their services. In addition to general conceptions of the firm and attempts to explain their constitution, there are numerous more specific conceptions of particular functions, focussing e.g. on manufacturing, finance, marketing etc. For Information Systems all of the above mentioned conceptions of the enterprise – as well as numerous others – can be relevant, if they contribute to specific aspects of designing and using information systems. However, a generic Information Systems perspective of the enterprise recommends accounting for information systems. Therefore, it makes sense to differentiate an enterprise into two constituent parts: the *action system* and the corresponding information system (see 3.2).

The term "action" denotes a core characteristic of human life – a characteristic that serves to distinguish our doing from that of animals. It is closely related to the cultural context and to language. Therefore, the term can hardly be defined in a comprehensive way. Instead, its interpretation relies in part on referring to our own experience as conscious actors. Due to the foundational function of actions for understanding human life and human interactions, it is not surprising that it is the core term of various so called action theories in Philosophy and Sociology ((Habermas 1984), (Parsons 1978), (Argyris 1985)). While a comprehensive consideration of respective approaches is not required for our work, they still have an impact on certain aspects of our conception.

> An **action** is characterized by an actor who performs it, an object it is related to and an intention that drives the actor.
>
> An **action system** is a system of interrelated actions that reflect the corresponding actors' intentions and abilities, organizational goals and guidelines, contextual threats and opportunities, as well as mutual expectations. An action system may consist of one actor only.

Beyond this ostensible description, the concept of an action system is accompanied by a number of subtle characteristics that have important implications on the appropriate development and use of information systems. Similar to information systems, language is a key element of action systems: An action system is based on communication and cooperation which in turn imply the existence of a common language. At the same time, actions enrich utterances with meaning and reproduce certain patterns of reducing complexity. In other words: They constitute and reproduce *sense*. "Action, perception, and sense-making exist in a circular, tightly coupled relationship ..." ((Weick 1979), p. 159). As a consequence, action systems will usually bulk against a formal specification. The concepts they are based on are often characterized by intentional semantics: The intentions that they reflect make sense only through references to the corresponding actors' "Lebenswelt" (literally: "life world") (Schütz 1981). Hence, describing action systems with formal languages only goes along with the risk of dysfunctional simplifications. From an epistemological point of view this suggests to not only focus on observable behavioural aspects, but to also consider a hermeneutic approach that aims at "empathy .. or re-creation in the mind of the scholar of the mental atmosphere, the thoughts and feelings and motivations, of the objects of his study." ((Wright 1971), p. 6)

Against this background, I propose a conceptualisation of action systems that is based on the following characteristics:

Intentional semantics and pragmatics: Action systems are not "there for the picking" as Bertrand Meyer suggests with respect to the discovery of objects. While this is arguably the case for most domains, it has a particular relevance for action systems. Action systems become accessible only through the language that describes and constitutes them. As a consequence, the analysis of action systems has to focus on corresponding descriptions and conversations. The term "domain of discourse", sometimes also called "universe of discourse" emphasizes this aspect.

Contingent subject: The involved actors will usually have different agendas, some of which are hidden. Furthermore, their actions are affected by assumptions about (mutual) expectations. While the idealized conception of enterprises assumes clear goals, objectivity and rational action, action systems are often "saturated with subjectivity, abstraction, guesses, … and arbitrariness" ((Weick 1979), p. 5). In other words: Action systems are characterized by multiple contingencies, which may be reciprocally intensified ((Luhmann 1984), p. 148 ff.).

Resistance to change: Often, the analysis of action systems is aimed at change, e.g. to improve efficiency, to decrease costs etc. However, action systems – and their linguistic foundations – constitute and reproduce sense, which is essential for understanding a complex environment and for reducing uncertainty and risk. As a consequence, action system will often show a remarkable persistence; many actors will be extremely reluctant to accept or even support change ((Morgan 1986), p. 233 ff., (O'Toole 1995)).

Penetrated by computer-supported information systems: To an ever growing extent enterprises are shaped by the use of information systems. In those companies where many stakeholders primarily interact with the information instead of people, there is even good cause for the pointed phrase "The information system is the enterprise".

Relevance of informal context: The success of action systems depends on individual intentions, motivations and commitment. Therefore, it is often not sufficient to account for "substantial" actions that are directly related to fulfilling certain tasks. Instead, "symbolic" actions may be required that are aimed at fostering motivation and commitment ((Pfeffer 1981), p. 5). Since, they are directly related to characteristics of a human actor such as charisma, persuasive power, empathy etc., they can hardly be reproduced by an information system.

5.2 Enterprise Models

Exploiting the potential of information systems will often require reorganizing existing patterns of action – sometimes in a radical way. Therefore, analysis and design of information systems should usually be done conjointly with analysing and designing the organisational action system. This consideration leads to the first, most generic conceptualisation of the term:

> An **enterprise model** integrates at least one conceptual model of an organisational action system with at least one conceptual model of a corresponding information system.

Note that action system and information system are not limited by the boundaries of a particular organisation. Instead, an enterprise model may represent inter-organisational action systems, too. It is essential that the models that constitute an enterprise model are integrated through the use of common concepts. This does not only foster the integrity of an enterprise model, it also provides a medium for users with different professional backgrounds to communicate more effectively: While everybody should have a good chance to find an abstraction that corresponds to his personal perception (and conception), he is also supported in realizing how the models he prefers are related to those of other stakeholders.

The complexity of both action system and information system demands for separation of concerns and professionalization. Professionalization goes along with specialized terminologies. Separation of concerns creates the need for coordination and, hence, for communication across different professional *perspectives*. The psychological concept of perspective relates to the concept of *perspectivity* ("Perspektivität" in German) which has a long tradition in Philosophy, Psychology and Sociology. It serves to express that the way an individual perceives and understands the world, i. e. his "Weltanschauung", is characterized by a specific perspective, i. e. a cognitive disposition that is shaped by socialisation, experiences, language games, etc. Hence, a perspective as a *psychological construct* constitutes a conception of reality, comparable to a particular viewpoint in spatial perception ((Graumann 1993), p. 159), which helps to reduce complexity by constituting sense ((Luhmann 1977), p. 182). According to Wollnik it is reflected in a certain use of language, in certain interests and intentions, hence in a particular pragmatic relevance(Wollnik 1986), p. 61). If perspectives are shared among individuals, they foster communication, otherwise they impede communication. The outstanding relevance, the psychological concept of perspective may have as a foundation for analysing and understanding social systems has insistently been characterized by Schütz:

> "Living in the world, we live with others and for others, orienting our lives to them. In experiencing them as *others*, as contemporaries and fellow creatures, as predecessors and successors, by joining with them in common activity and work, influencing them and being influenced by them in turn – in doing all these things we *understand* the behaviour of others and assume that they understand us. In these acts of establishing or interpreting meanings there is built up for us in varying degrees of anonymity, in greater or lesser intimacy of experience, in manifold intersecting perspectives, the structural meaning of the social world, which is as much our world (strictly speaking, my world) at the world of the others." (translated from (Schütz 1981), p. 17)

To express the support of various perspectives on the enterprise, the term "multi-perspective enterprise model" has been coined.

> A **multi-perspective enterprise model** is an enterprise model that emphasizes accounting for *perspectives*. The term "multi-perspective" is purposefully overloaded. On the one hand, it represents different conceptions of perspective. On the other hand, it refers

to differentiating specific perspectives related to one conception of perspective. The first conception is a psychological one: In this conception, a perspective represents a specific professional background that corresponds to cognitive dispositions, technical languages, specific goals and capabilities of prospective users. The second conception refers to the *representation* of perspectives within an enterprise model. In this sense it relates directly to the respective DSML that are expected to provide concepts that correspond to those characteristic for certain (psychological) perspectives.

To promote semantic equivalence of concepts that are shared by different DSML and, hence, to allow for a tight integration of respective models, all DSML that are part of an enterprise modelling method should be specified with the same meta modelling language. A meta modelling language and the corresponding set of DSML form a *language architecture*.

An enterprise model should be accessible by software and by humans. The first case implies a machine readable representation. The second case requires representations that are comprehensible for humans. Often, diagrams will be the representation of choice – based on the assumption that graphical representations are especially suited for reducing complexity and fostering comprehensibility in cases of multiple mutual relationships. Nevertheless, it can be appropriate for certain parts of an enterprise model to choose a textual representation. The complexity of the subject demands for covering different levels of abstraction and detail. Usually, approaches to enterprise modelling include a conceptual framework to structure an enterprise into essential aspects (for example: (Scheer 2001), (Zachman 1987)). Such a "ballpark view" allows developing a common, high-level conception of an enterprise, which serves as a starting point for further analysis. After a discussion on the "ballpark view" level resulted in the identification of problem areas, these can be further analysed by developing more detailed models using corresponding DSML. These models can either represent the current state of an enterprise or a possible future state. The various models that form an enterprise model are represented by corresponding diagrams. Figure 11 shows an example of a high level model of an enterprise that is associated with various diagrams which are representations of models specified with DSML.

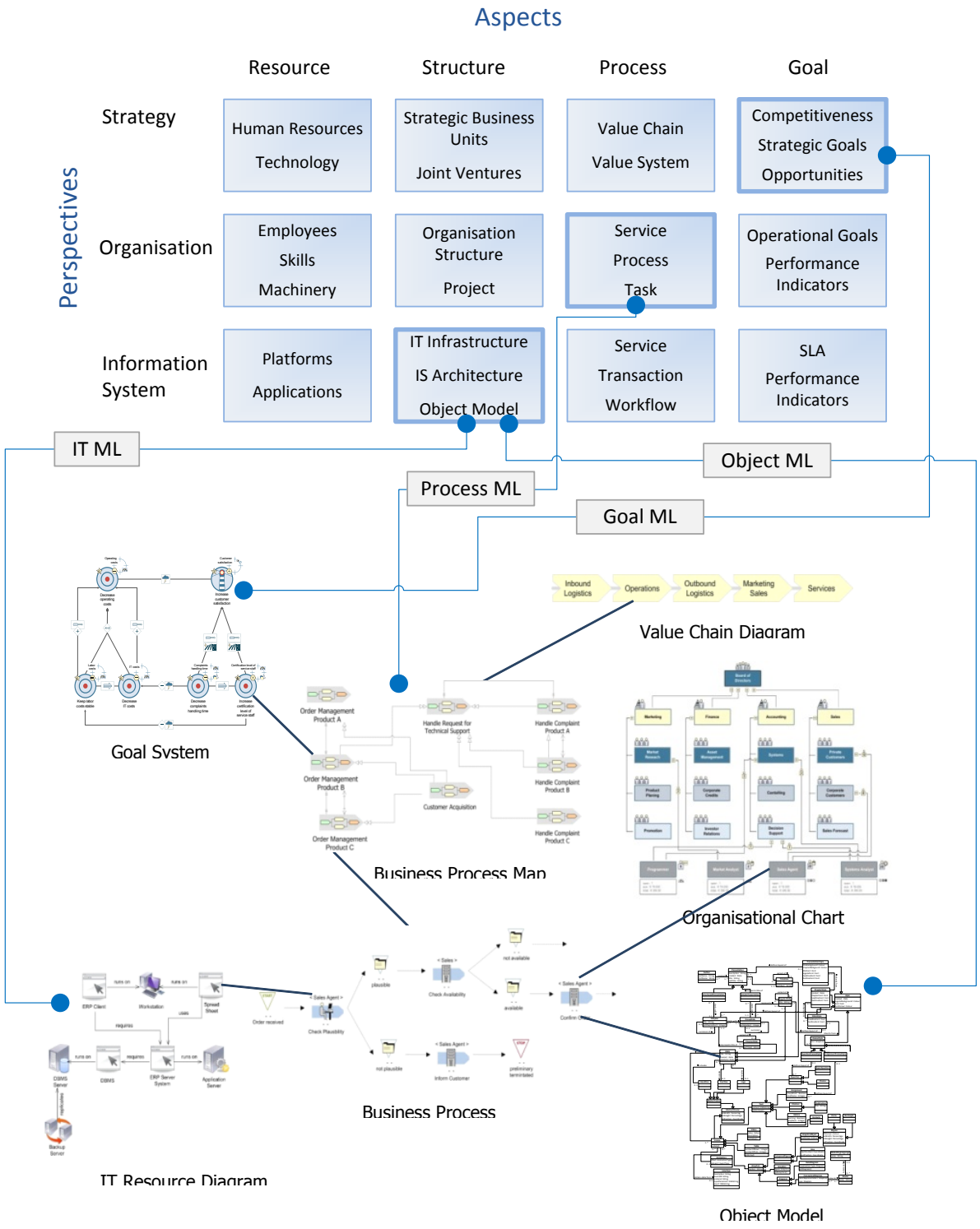

Figure 11: Exemplary Representation of an Enterprise Model

The term „multi-perspective enterprise model“ may be taken one step further by stressing an additional, critical *meta perspectives*. It recommends supplementing – and confronting – an enterprise model with the relevant context, i. e. with aspects that are required for a reflective interpretation of an enterprise model but that bulk against formalisation. In other words, it suggests enhancing the engineering approach that is promoted by the use of (semi-) formal

DSML to design systems, by a reflective, hermeneutic perspective that accounts for the limitations of enterprise models – especially with respect to subtle aspects of organisational psychology such as hidden agendas, opportunistic action, politics, symbolic action, corporate culture etc. It recommends accounting for interaction, empathy, introspection to gain an elaborate appreciation of organisations in general, of a specific social setting in a particular case. With respect to promoting awareness of and discourses about these informal aspects, they need to be represented in an appropriate form that is suited to supplement an enterprise model. Examples for possible approaches are the use of images (Morgan 1986), the use of "rich pictures" or other instruments suggested by the Soft Systems Methodology ((Checkland and Scholes 1999).

5.3 Enterprise Architecture

In recent years, the term "enterprise architecture" has gained remarkable attention. As the following definitions show, it is closely related to the term "enterprise model".

> "A coherent set of descriptions, covering a regulations-oriented, design-oriented and patterns-oriented perspective on an enterprise, which provides indicators and controls that enable the informed governance of the enterprise's evolution and success." ((Land, Waage et al. 2009), p. 34)
>
> "EA is coherent whole of principles, methods and models that are used in the design and realization of an enterprises organizational structure, business processes, information systems, and infrastructure." ((Lankhorst 2005), p. 3)

These definitions as well as others, e.g. (Urbaczewski and Mrdalj 2006), (Foorthuis and Brinkkemper 2007), do not point at substantial differences to enterprise models – except for the fact that Lankhorst subsumes methods under the term, too. Often, the use of the term "enterprise architecture" is accompanied by the term "enterprise architecture management". It stresses the need for a specialised approach to manage the creation and use of enterprise architectures:

> "EA management is a continuous and self maintaining management function seeking to improve the alignment of business and IT and to guide the managed evolution of an (virtual) enterprise. Based on a holistic perspective on the enterprise furnished with information from other enterprise-level management functions it provides input to, exerts control over, and defines guidelines for these enterprise-level management functions. The EA management function consists of the activities develop & describe, communicate & enact, and analyze & evaluate." ((Buckl, Matthes et al. 2010), p. 152)

Methods for enterprise modelling are usually also aimed at covering the entire lifecycle of an enterprise model. Therefore, the differences between enterprise model and enterprise architecture are mainly related to the intended audience. Enterprise modelling is aimed at various groups of stakeholders that are involved in planning, implementing, using and maintaining information systems. Therefore, enterprise models are supposed to offer a variety of corresponding abstractions. These include models that serve as a foundation of software devel-

opment. Therefore, the development of respective DSML is a particular characteristic of enterprise modelling. Different from that, enterprise architecture mainly targets IT management. Therefore, it puts less emphasis on the specification of DSML. If DSML are included in an approach to enterprise architecture, they will usually not account for implementation issues. In any case, there is no fundamental difference between both approaches. Instead, one can regard the abstractions focussed by enterprise architectures as an integral part of a more comprehensive enterprise model.

6 Concluding Remarks

This report is aimed at a proposing the terminological foundation for enterprise modelling. In this sense, the concepts presented in this report are intended to provide readers with a framework that helps with getting a better understanding of conceptual modelling in general, of enterprise modelling in particular. It is also aimed at contributing to the development of a common terminological foundation that fosters communication and collaboration. I would hope that the suggestions made in this report meet this intention to some extent. However, not all concepts presented in this report are suited for everybody who is interested in enterprise modelling. Therefore, some concepts are differentiated into basic versions which should be appropriate for novice users and more elaborate versions. While conceptualising key terms is an important prerequisite for scientific studies, it is not restricted to the beginning of a research process – like it is the case for building the foundation of a house. Instead, the process of analysing a subject is characterized by a continuous reflection of key terms – and their occasional adaptation. Hence, a terminology is both, an instrument for and a result of analysis. This is even more the case, when we consider essential terms such as “model”, “language” or “domain”, since it is hard – if not impossible – to reduce them to other concepts. At the same time, they represent complex constructs that cannot be clarified by simply pointing at the obvious. Therefore, this report is also intended as a contribution to an advanced discourse on the terminological foundation of our discipline. This thought corresponds to a more general epistemological consideration. Language is a mandatory tool for thinking. It helps building on proven ways of perceiving and conceptualising problem domains. However, at the same time, it may hamper our creativity by promoting certain kinds of perception and conceptualisation. This is a dilemma that we can hardly escape. We can only try to be oblivious of it: On the one hand, by refining existing terminologies and corresponding world views; on the other hand, by occasionally leaning back and trying to go beyond the limitations of the concepts that seem constitutional for our way of thinking.

References

Alpar, P., R. Alt, et al. (2011). Anwendungsorientierte Wirtschaftsinformatik. Strategische Planung, Entwicklung und Nutzung von Informations- und Kommunikationssystemen. Wiesbaden, Vieweg.

Argyris, C. (1985). Action Science, Concepts, Methods, and Skills for Research and Intervention. San Francisco, Jossey-Bass.

Barnard, C. I. (1938). The Function o f the Executive. Cambridge.

Berger, P. L. and T. Luckmann (1966). The Social Construction of Reality: A Treatise in the Sociology of Knowledge. Garden City, N.Y., Doubleday.

Bernus, P. and G. Schmidt (2006). What is an Information System? Handbook on Architectures of Information Systems. P. Bernus, K. Mertins and G. Schmidt. Berlin, Springer: 1-9.

Brynjolfsson, E. (1993). "The productivity paradox of information technology." Communications of the ACM 36(12): 66–77.

Buckl, S., F. Matthes, et al. (2010). A Conceptual Framework for Enterprise Architecture Design. Trends in Enterprise Architecture Research. E. Proper, M. Lankhorst, M. Schönherr, J. Barjis and S. Overbeek, Springer Berlin Heidelberg. 70: 44-56.

Bunge, M. A. (1977). Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World. Dordrecht, Boston, Reidel.

Carr, N. G. (2004). Does IT Matter? Information Technology and the Corrosion of Competitive Advantage. Cambridge, MA, Harvard Business School Press.

Chaffey, D. and S. Wood (2005). Business Information Management. Improving Performance Using Information Systems. Harlow, London, Prentice Hall.

Checkland, P. and J. Scholes (1999). Soft Systems Methodology in Action. New York, Wiley.

Falkenberg, E. D., W. Hesse, et al. (1998). A Framework of Information System Concepts. The FRISCO Report, IFIP.

Ferstl, O. K. and E. J. Sinz (2005). Grundlagen der Wirtschaftsinformatik. München, Wien, Oldenbourg.

Floyd, C. (1992). Human Questions in Computer Science. Software Development and Reality Construction. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slawik. Berlin et al.: 15-27.

Foorthuis, R. M. and S. Brinkkemper (2007). A Framework for Project Architecture in the Context of Enterprise Architecture. Proceedings of the TEAR Workshop 2007: 51–60.

Frank, U. (2000a). "Delegation: An Important Concept for the Appropriate Design of Object Models." Journal of Object-Oriented Programming 13(3): 13-18.

Frank, U. (2000b). "An Important Concept for the Appropriate Design of Object Models." Journal of Object-Oriented Programming 13(3): 13-18.

Frank, U. (2001). "A Conceptual Foundation for Versatile E-Commerce Platforms." Journal of Electronic Commerce Research 2(2).

Frank, U. (2008). Integration - Reflections on a Pivotal Concept for Designing and Evaluating Information Systems. Information Systems and e-Business Technologies. R. Kaschek, C. Kop, C. Steinberger and G. Fliedl. Berlin, Springer. 5: 11-22.

Frank, U. (2010). Outline of a Method for Designing Domain-Specific Modelling Languages. ICB Research Report University Duisburg-Essen. No. 42.

Frank, U. (2011). Multi-Perspective Enterprise Modelling: Background and Terminological Foundation. ICB Research Report University Duisburg-Essen. No. 45.

Frank, U. (2012). Specialisation in Business Process Modelling: Motivation, Approaches and Limitations. ICB-Research Report, University Duisburg-Essen. 51.

Frank, U. and S. Strecker (2009). Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems. ICB Research Report, No. 31, University Duisburg-Essen.

Glasersfeld, E. v. (1995). Radical Constructivism: A Way of Knowing and Learning. London, Falmer Press.

Graumann, C.-F. (1993). "Perspektivität in Kognition und Sprache." SPIEL: Siegener Periodicum zur Internationalen Empirischen Literaturwissenschaft 12(2): 156-172.

Grossmann, R. (1983). The Categorial Structure of the World. Bloomington, Indiana University Press.

Gruber, T. R. (1992). A Translation Approach to Portable Ontology Specifications. Technical Report KSL 92-71. S. U. Computer Science Department.

Gutenberg, E. (1929). Die Unternehmung als Gegenstand betriebswirtschaftlicher Theorie. Berlin, Wien.

Habermas, J. (1984). Theorie des kommunikativen Handelns. Handlungsrationalität und gesellschaftliche Rationalisierung. Frankfurt/M., Suhrkamp.

Heinrich, L. J., A. Heinzl, et al. (2010). Wirtschaftsinformatik: Einführung und Grundlegung. Berlin, Heidelberg, Springer.

Henderson-Sellers, B. (2010). "Situational Method Engineering: State-of-the-Art Review." Journal of Universal Computer Science 16(3): 424-478.

Henderson-Sellers, B. (2011). Random Thoughts on Multi-level Conceptual Modelling. The Evolution of Conceptual Modeling. From a Historical Perspective towards the Future of Conceptual Modeling. R. Kaschek and L. Delcambre. Berlin, Heidelberg, Springer: 93–116.

Kant, I. (1976). Kritik der reinen Vernunft. Frankfurt/M., Suhrkamp.

Klein, H. K. and K. Lyytinen (1992). Towards a New Understanding of Data Modelling. Software Development as Reality Construction. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slavik. Berlin et al.: 86-100.

Künne, W. (2003). Conceptions of Truth. Oxford, Oxford University Press.

Land, M. O. t., M. Waage, et al. (2009). Enterprise architecture: Creating value by informed governance. Berlin, Springer.

Lankhorst, M. (2005). Enterprise Architecture at Work. Modelling, Communiation and Analysis. Berlin, Heidelberg, New York, Springer.

Laudon, K. C. and J. P. Laudon (2005). Essentials of Management Information Systems. Managing the Digital Firm. Upper Saddle River, NJ, Pearson/Prentice Hall.

Luhmann, N. (1977). Zweckbegriff und Systemrationalität. Frankfurt/M., Suhrkamp.

Luhmann, N. (1984). Soziale Systeme. Grundriß einer allgemeinen Theorie. Frankfurt/M., Suhrkamp.

Mahr, B. (2009). "Die Informatik und die Logik der Modelle." Informatik Spektrum 32(3): 228-249.

Mata, F. J., W. L. Fuerst, et al. (1995). "Information Technology and Sustained Competitive Advantage: A Resource-Based Analysis." MIS Quarterly 19(4): 487-505.

Maturana, H. R. (1987). Kognition. Der Diskurs des Radikalen Konstruktivismus. S. J. Schmidt. Frankfurt/M., Suhrkamp: 89-118.

Maturana, H. R. and F. J. Varela (1987). The Tree of Knowledge: The Biological Roots of Human Understanding. Boston, New Science Library.

Meyer, B. (1997). Object-Oriented Software Construction, Prentice Hall International.

Morgan, G. (1986). Images of Organization. Thousands Oaks, London, New Delhi, Sage.

Mylopoulos, J. and H. J. Levesque (1984). An Overview of Knowledge Representation. On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming. M. L. Brodie, J. Mylopoulos and J. W. Schmidt. Berlin

Heidelberg, Springer: 3-17.

O'Toole, J. (1995). Leading Change. San Francisco, Jossey-Bass.

Olivé, A. (2007). Conceptual Modeling of Information Systmes. Berlin, Heidelberg, New York, Springer.

Ortner, E. (1997). Methodenneutraler Fachentwurf. Stuttgart, Leipzig, Teubner.

Parsons, T. (1978). Action Theory and the Human Condition. New York, Free Press.

Pfeffer, J. (1981). Management as as Symbolic Action. The Creation and Maintenance of Organizational Paradigms. Research in Organizational Behavior. L. L. Cummings and B. M. Staw. Greenwich, London. 3.

Porter, M. E. and V. E. Millar (1985). "How Information Gives You Competitive Advantage." Harvard Business Review 63(4): 149-160.

Scheer, A. W. (2001). ARIS - Modellierungsmethoden, Metamodelle, Anwendungen. Berlin, Springer.

Schütte, R. (2000). Zum Realitätsbezug von Informationsmodellen. U. D.-E. Institut für Produktion und Industrielles Informationsmanagement. Essen.

Schütz, A. (1981). Der sinnhafte Aufbau der sozialen Welt. Frankfurt/M., Suhrkamp.

Schwarzer, B. and H. Krcmar (2004). Wirtschaftsinformatik. Stuttgart, Schäffer Poeschel.

Schwemmer, O. (1984). Ontologie. Enzyklopädie Philosophie und Wissenschaftstheorie. J. Mittelstraß. Mannheim, B.I. Wissenschaftsverlag. 2: 1077-1079.

Shannon, C. E. (1948). "A Mathematical Theory of Communication." The Bell System Technical Journal 27: 379-423.

Simon, H. A. (1949). Administrative Behavior. A Study o f Decision-Making Processes in Administrative Organization. New York.

Stachowiak, H. (1973). Allgemeine Modelltheorie. Wien, Springer.

Stachowiak, H. (1983). Konstruierte Wirklichkeit. Modelle – Konstruktion der Wirklichkeit. H. Stachowiak. München: 10-16.

Stegmüller, W. (1986). Kripkes Deutung der Spätphilosophie Wittgensteins. Kommentarversuch über einen versuchten Kommentar. Stuttgart.

Urbaczewski, L. and S. Mrdalj (2006). "A comparison of enterprise architecture frameworks." Issues in Information Systems 7(2): 18–23.

Weick, K. E. (1979). The Social Psychology of Organizing. New York, McGraw-Hill.

Williamson, O. E. (1985). The Economic Institutions of Capitalism, Firms, Markets, Relational Contracting. New York.

Wittgenstein, L. (1981). Tractatus Logico-Philosophicus. Englewood Cliffs, NJ, Routledge Kegan Paul.

Wittgenstein, L. (2001). Philosophical Investigations. The German Text, with a Revised English Translation. Malden, MA; Oxford: Carlton, Blackwell.

Wittmann, W. (1959). Unternehmung und unvollkommene Information. Köln, Opladen.

Wollnik, M. (1986). Implementierung computergestützter Informationssysteme. Berlin, New York.

Wolters, G. (1984). Modell. Enzyklopädie Philosophie und Wissenschaftstheorie. J. Mittelstraß. Mannheim, Wien

Zürich, Bibliographisches Institut. 2: 911-913.

Wright, G. H. v. (1971). Explanation and Understanding. Ithaca, NY, Cornell University Press.

Wright, G. H. v. (1974). Erklären und Verstehen. Frankfurt/M., Athenäum Fischer.

Zachman, J. A. (1987). "A framework for information systems architecture." IBM Systems Journal 26(3): 276–292.

Zelewski, S. (1995). Petrinetzbasierte Modellierung komplexer Produktionssysteme. Arbeitsbericht des Instituts für Produktionswirtschaft und industrielle Informationswirtschaft. Essen, Universität Leipzig

Previously published ICB - Research Reports

2011

No 45 (November 2011)

Frank, Ulrich; Strecker, Stefan; Heise, David; Kattenstroth, Heiko; Schauer, Carola: "Leitfaden zur Erstellung wissenschaftlicher Arbeiten in der Wirtschaftsinformatik"

No 44 (September 2010)

Berenbach, Brian; Daneva, Maya; Dörr, Jörg; Frickler, Samuel; Gervasi, Vincenzo; Glinz, Martin; Herrmann, Andrea; Krams, Benedikt; Madhavji, Nazim H.; Paech, Barbara; Schockert, Sixten; Seyff, Norbert (Eds): "17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2011). Proceedings of the REFSQ 2011 Workshops REEW, EPICAL and RePriCo, the REFSQ 2011 Empirical Track (Empirical Live Experiment and Empirical Research Fair), and the REFSQ 2011 Doctoral Symposium"

No 43 (February 2011)

Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Lnguage Architecture – 2nd Edition"

2010

No 42 (December)

Frank, Ulrich: "Outline of a Method for Designing Domain-Specific Modelling Languages"

No 41 (December)

Adelsberger,Heimo; Drechsler, Andreas (Eds): "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive – Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen"

No 40 (October 2010)

Bürsner, Simone; Dörr, Jörg; Gehlert, Andreas; Herrmann, Andrea; Herzwurm, Georg; Janzen, Dirk; Merten, Thorsten; Pietsch, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar (Eds): "16th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings of the Workshops CreaRE, PLREQ, RePriCo and RESC"

No 39 (May 2010)

Strecker, Stefan; Heise, David; Frank, Ulrich: "Entwurf einer Mentoring-Konzeption für den Studiengang M.Sc. Wirtschaftsinformatik an der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen"

No 38 (February 2010)

Schauer, Carola: "Wie praxisorientiert ist die Wirtschaftsinformatik? Einschätzungen von CIOs und WI-Professoren"

No 37 (January 2010)

Benavides, David; Batory, Don; Grunbacher, Paul (Eds.): "Fourth International Workshop on Variability Modelling of Software-intensive Systems"

2009

No 36 (December 2009)
Strecker, Stefan: "Ein Kommentar zur Diskussion um Begriff und Verständnis der IT-Governance - Anregungen zu einer kritischen Reflexion"

No 35 (August 2009)
Rüngeler, Irene; Tüxen, Michael; Rathgeb, Erwin P.:"Considerations on Handling Link Errors in STCP"

No 34 (June 2009)
Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaption and Beyond"

No 33 (May 2009)
Adelsberger,Heimo; Drechsler , Andreas; Bruckmann, Tobias; Kalvelage, Peter; Kinne, Sophia; Pellinger, Jan; Rosenberger, Marcel; Trepper, Tobias: „Einsatz von Social Software in Unternehmen – Studie über Umfang und Zweck der Nutzung"

No 32 (April 2009)
Barth, Manfred; Gadatsch, Andreas; Kütz, Martin; Rüding, Otto; Schauer, Hanno; Strecker, Stefan: „Leitbild IT-Controller/-in – Beitrag der Fachgruppe IT-Controlling der Gesellschaft für Informatik e. V."

No 31 (April 2009)
Frank, Ulrich; Strecker, Stefan: "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems – Requirements, Conceptual Foundation and Design Options"

No 30 (February 2009)
Schauer, Hanno; Wolff, Frank: „Kriterien guter Wissensarbeit – Ein Vorschlag aus dem Blickwinkel der Wissenschaftstheorie (Langfassung)"

No 29 (January 2009)
Benavides, David; Metzger, Andreas; Eisenecker, Ulrich (Eds.): "Third International Workshop on Variability Modelling of Software-intensive Systems"

2008

No 28 (December 2008)
Goedicke, Michael; Striewe, Michael; Balz, Moritz: „Computer Aided Assessments and Programming Exercises with JACK"

No 27 (December 2008)
Schauer, Carola: "Größe und Ausrichtung der Disziplin Wirtschaftsinformatik an Universitäten im deutschsprachigen Raum - Aktueller Status und Entwicklung seit 1992"

No 26 (September 2008)
Milen, Tilev; Bruno Müller-Clostermann: " CapSys: A Tool for Macroscopic Capacity Planning"

No 25 (August 2008)
Eicker, Stefan; Spies, Thorsten; Tschersich, Markus: "Einsatz von Multi-Touch beim Softwaredesign am Beispiel der CRC Card-Methode"

No 24 (August 2008)
Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Language Architecture – Revised Version"

No 23 (January 2008)
Sprenger, Jonas; Jung, Jürgen: "Enterprise Modelling in the Context of Manufacturing – Outline of an Approach Supporting Production Planning"

No 22 (January 2008)
Heymans, Patrick; Kang, Kyo-Chul; Metzger, Andreas, Pohl, Klaus (Eds.): "Second International Workshop on Variability Modelling of Software-intensive Systems"

2007

No 21 (September 2007)
Eicker, Stefan; Annett Nagel; Peter M. Schuler: "Flexibilität im Geschäftsprozess-management-Kreislauf"

No 20 (August 2007)
Blau, Holger; Eicker, Stefan; Spies, Thorsten: "Reifegradüberwachung von Software"

No 19 (June 2007)
Schauer, Carola: "Relevance and Success of IS Teaching and Research: An Analysis of the ‚Relevance Debate'

No 18 (May 2007)
Schauer, Carola: "Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre"

No 17 (May 2007)
Schauer, Carola; Schmeing, Tobias: "Development of IS Teaching in North-America: An Analysis of Model Curricula"

No 16 (May 2007)
Müller-Clostermann, Bruno; Tilev, Milen: "Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning"

No 15 (April 2007)
Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals – Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"

No 14 (March 2007)
Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"

No 13 (February 2007)
Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"

No 12 (February 2007)
Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"

No 11 (February 2007)
Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT-Managements – Grundlagen, Anforderungen und Metamodell"

No 10 (February 2007)
Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"

No 9 (February 2007)
Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"

No 8 (February 2007)
Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"

2006

No 7 (December 2006)
Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"

No 6 (April 2006)
Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten – Ein Diskussionsbeitrag"

No 5 (April 2006)
Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"

No 4 (February 2006)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III – Results Wirtschaftsinformatik Discipline"

2005

No 3 (December 2005)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II – Results Information Systems Discipline"

No 2 (December 2005)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I – Research Objectives and Method"

No 1 (August 2005)
Lange, Carola: „Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"

Research Group	Core Research Topics
Prof. Dr. H. H. Adelsberger Information Systems for Production and Operations Management	E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr. F.-D. Dorloff Procurement, Logistics and Information Management	E-Business, E-Procurement, E-Government
Prof. Dr. K. Echtle Dependability of Computing Systems	Dependability of Computing Systems
Prof. Dr. S. Eicker Information Systems and Software Engineering	Process Models, Software-Architectures
Prof. Dr. U. Frank Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
Prof. Dr. M. Goedicke Specification of Software Systems	Distributed Systems, Software Components, CSCW
Prof. Dr. V. Gruhn Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
PD Dr. C. Klüver Computer Based Analysis of Social Complexity	Soft Computing, Modeling of Social, Cognitive, and Economic Processes, Development of Algorithms
Prof. Dr. T. Kollmann E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
Prof. Dr. B. Müller-Clostermann Systems Modelling	Performance Evaluation of Computer and Communication Systems, Modelling and Simulation
Prof. Dr. K. Pohl Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
Prof. Dr.-Ing. E. Rathgeb Computer Networking Technology	Computer Networking Technology
Prof. Dr. E. Rukzio Mobile Mensch Computer Interaktion mit Software Services	Novel Interaction Technologies, Personal Projectors, Pervasive User Interfaces, Ubiquitous Computing
Prof. Dr. R. Unland Data Management Systems and Knowledge Representation	Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching
Prof. Dr. S. Zelewski Institute of Production and Industrial Information Management	Industrial Business Processes, Innovation Management, Information Management, Economic Analyses

For more information visit us on the Web: **http://www.icb.uni-due.de**

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)